

Universidade do Sul de Santa Catarina

Banco de Dados II

Disciplina na modalidade a distância

Palhoça

UnisulVirtual

2006

Créditos

Unisul - Universidade do Sul de Santa Catarina

UnisulVirtual - Educação Superior a Distância

Campus UnisulVirtual

Avenida dos Lagos, 41
Cidade Universitária Pedra Branca
Palhoça – SC - 88137-100
Fone/fax: (48) 3279-1242 e
3279-1271
E-mail: cursovirtual@unisul.br
Site: www.virtual.unisul.br

Reitor Unisul

Gerson Luiz Joner da Silveira

Vice-Reitor e Pró-Reitor Acadêmico

Sebastião Salésio Heerd

Chefe de Gabinete da Reitoria

Fabian Martins de Castro

Pró-Reitor Administrativo

Marcus Vinícius Anátoles da Silva
Ferreira

Campus Sul

Diretor: Valter Alves Schmitz Neto
Diretora adjunta: Alexandra
Orsoni

Campus Norte

Diretor: Ailton Nazareno Soares
Diretora adjunta: Cibele Schuelter

Campus UnisulVirtual

Diretor: João Vianney
Diretora adjunta: Jucimara
Roesler

Equipe UnisulVirtual

Administração

Renato André Luz
Valmir Venício Inácio

Avaliação Institucional

Dênia Falcão de Bittencourt

Biblioteca

Soraya Arruda Waltrick

Capacitação e Apoio Pedagógico à Tutoria

Angelita Marçal Flores
(Coordenadora)
Caroline Batista
Enzo de Oliveira Moreira
Patrícia Meneghel
Vanessa Francine Corrêa

Coordenação dos Cursos

Adriano Sérgio da Cunha
Aloísio José Rodrigues
Ana Luisa Mülbart
Ana Paula Reusing Pacheco
Charles Cesconetto
Diva Marília Flemming
Fabiano Ceretta
Itamar Pedro Bevilacqua
Janete Elza Felisbino
Jucimara Roesler
Lauro José Ballock
Lívia da Cruz (Auxiliar)
Luiz Guilherme Buchmann
Figueiredo
Luiz Otávio Botelho Lento
Marcelo Cavalcanti
Maria da Graça Poyer
Maria de Fátima Martins
(Auxiliar)
Mauro Faccioni Filho
Michelle D. Durieux Lopes Destri
Moacir Fogaça
Moacir Heerd
Nélio Herzmann
Onei Tadeu Dutra
Patrícia Alberton
Raulino Jacó Brüning
Rodrigo Nunes Lunardelli
Simone Andréa de Castilho
(Auxiliar)

Criação e Reconhecimento de Cursos

Diane Dal Mago
Vanderlei Brasil

Desenho Educacional

Design Instrucional

Daniela Erani Monteiro Will
(Coordenadora)
Carmen Maria Cipriani Pandini
Carolina Hoeller da Silva Boeing
Flávia Lumi Matuzawa
Karla Leonora Dahse Nunes
Leandro Kingeski Pacheco
Ligia Maria Soufen Tumolo
Márcia Loch
Viviane Bastos
Viviani Poyer

Acessibilidade

Vanessa de Andrade Manoel

Avaliação da Aprendizagem

Márcia Loch (Coordenadora)
Cristina Klipp de Oliveira
Silvana Denise Guimarães

Design Gráfico

Cristiano Neri Gonçalves Ribeiro
(Coordenador)
Adriana Ferreira dos Santos
Alex Sandro Xavier
Evandro Guedes Machado
Fernando Roberto Dias
Zimmermann
Higor Ghisi Luciano
Pedro Paulo Alves Teixeira
Rafael Pessi
Wilson Martins Filho

Disciplinas a Distância

Tade-Ane de Amorim
Cátia Melissa Rodrigues

Gerência Acadêmica

Patrícia Alberton

Gerência de Ensino

Ana Paula Reusing Pacheco

Logística de Encontros Presenciais

Márcia Luz de Oliveira
(Coordenadora)
Aracelli Araldi
Graciele Marinês Lindenmayr
Leticia Cristina Barbosa
Kênia Alexandra Costa Hermann
Priscila Santos Alves

Formatura e Eventos

Jackson Schuelter Wiggers

Logística de Materiais

Jeferson Cassiano Almeida da
Costa (Coordenador)
José Carlos Teixeira
Eduardo Kraus

Monitoria e Suporte

Rafael da Cunha Lara
(coordenador)
Adriana Silveira
Andréia Drewes
Caroline Mendonça
Cristiano Dalazen
Dyego Rachadel
Edison Rodrigo Valim
Francielle Arruda
Gabriela Malinverni Barbieri
Jonatas Collaço de Souza
Josiane Conceição Leal
Maria Eugênia Ferreira Celeghin
Rachel Lopes C. Pinto
Vinícius Maykot Serafim

Produção Industrial e Suporte

Arthur Emmanuel F. Silveira
(coordenador)
Francisco Asp

Relacionamento com o Mercado

Walter Félix Cardoso Júnior

Secretaria de Ensino a Distância

Karine Augusta Zanoni
Albuquerque
(Secretária de ensino)
Ana Paula Pereira
Andréa Luci Mandira
Carla Cristina Sbardella
Deise Marcelo Antunes
Djeime Sammer Bortolotti
Franciele da Silva Bruchado
Grasiela Martins
James Marcel Silva Ribeiro
Jenniffer Camargo
Lamuniê Souza
Lauana de Lima Bezerra
Liana Pamplona
Marcelo José Soares
Marcos Alcides Medeiros Junior
Maria Isabel Aragon
Olavo Lajús
Priscilla Geovana Pagani
Rosângela Mara Siegel
Silvana Henrique Silva
Vanilda Liordina Heerd
Vilmar Isaurino Vidal

Secretária Executiva

Viviane Schalata Martins

Tecnologia

Osmar de Oliveira Braz Júnior
(Coordenador)
Jefferson Amorin Oliveira
Ricardo Alexandre Bianchini

Apresentação

Este livro didático corresponde à disciplina **Banco de Dados II**.

O material foi elaborado visando a uma aprendizagem autônoma, abordando conteúdos especialmente selecionados e adotando uma linguagem que facilite seu estudo a distância.

Por falar em distância, isso não significa que você estará sozinho. Não esqueça que sua caminhada nesta disciplina será acompanhada constantemente pelo Sistema Tutorial da UnisulVirtual. Entre em contato sempre que sentir necessidade, seja por correio postal, fax, telefone, e-mail ou Espaço UnisulVirtual de Aprendizagem. Nossa equipe terá o maior prazer em atendê-lo, pois sua aprendizagem é nosso principal objetivo.

Bom estudo e sucesso!

Equipe UnisulVirtual.

Marcelo Medeiros

Banco de Dados II

Livro didático

Design instrucional

Flavia Lumi Matuzawa

Palhoça

UnisulVirtual

2006

Edição – Livro Didático

Professor Conteudista

Marcelo Medeiros

Design Instrucional

Flavia Lumi Matuzawa

ISBN 8560694-15-3

ISBN 978-85-60694-15-0

Projeto Gráfico e Capa

Equipe UnisulVirtual

Diagramação

Rafael Pessi

Revisão Ortográfica

B2B

005.75

M43 Medeiros, Marcelo

Banco de dados II : livro didático / Marcelo Medeiros ; design Instrucional Flavia Lumi Matuzawa, Viviane Bastos. – Palhoça : UnisulVirtual, 2006.

238 p. : il. ; 28 cm.

Inclui bibliografia.

ISBN 8560694-15-3

ISBN 978-85-60694-15-0

1. Banco de dados. 2. Banco de dados orientado a objetos. 3. Banco de dados relacionais. I. Matuzawa, Flavia Lumi. II. Bastos, Viviane. III. Título.

Sumário

Apresentação	03
Palavras do professor	09
Plano de estudo	11
UNIDADE 1 – Sistema computacional como solução do problema	15
UNIDADE 2 – Preparando o ambiente de banco de dados	43
UNIDADE 3 – Ambiente de programação Java	77
UNIDADE 4 – Linguagem de definição de dados - DDL	91
UNIDADE 5 – Ambiente de desenvolvimento integrado	107
UNIDADE 6 – Comandos em Java para conexão com banco de dados	127
UNIDADE 7 – Comandos em Java para Apresentação dos Dados	157
UNIDADE 8 – Criando a aplicação de cadastro de alunos	177
UNIDADE 9 – Criando as consultas para a Tabela de Alunos.....	209
Para concluir o estudo	225
Sobre o professor conteudista.....	227
Referências	229
Respostas e comentários das atividades de auto-avaliação	231

Palavras do professor



Olá, você está iniciando o estudo da disciplina de Banco de Dados II.

Se você parar um pouco para avaliar o seu dia-a-dia, você notará que convive com um mundo de informações, que são usadas nas suas atividades diárias, por mais simples que sejam.

Nesta disciplina você conhecerá sobre as formas de armazenamento e acesso aos dados gravados no computador e sobre as técnicas existentes que possibilitarão a você integrar um sistema de banco de dados e uma linguagem de programação, ou de forma resumida, criar um sistema computacional com acesso a banco de dados.

Esta disciplina possibilitará que você entenda melhor as regras de implementação que permitem que um sistema de banco de dados seja acessado e manipulado por um programa desenvolvido em uma linguagem de programação de alto nível, neste caso o Java.

Assim, espero que você possa aproveitar ao máximo o conteúdo desta disciplina.

Bom estudo!

Prof. Marcelo Medeiros



Plano de estudo

O plano de estudos visa a orientá-lo/a no desenvolvimento da Disciplina. Nele, você encontrará elementos que esclarecerão o contexto da Disciplina e sugerirão formas de organizar o seu tempo de estudos.

O processo de ensino e aprendizagem na UnisulVirtual leva em conta instrumentos que se articulam e se complementam. Assim, a construção de competências se dá sobre a articulação de metodologias e por meio das diversas formas de ação/ mediação.

São elementos desse processo:

- o livro didático;
- o Espaço UnisulVirtual de Aprendizagem - **EVA**;
- as atividades de avaliação (complementares, a distância e presenciais);
- o Sistema Tutorial.

Ementa

Fundamentação teórica do modelo relacional: restrições de integridade básica, integridade semântica. Mapeamento do modelo conceitual para o modelo relacional. Linguagem de consulta estruturada (SQL). Normalização. Banco de dados aplicados em sistemas web. Análise, projeto e implementação de banco de exemplos reais.

Objetivo

Apresentar os principais conceitos de sistemas de Banco de Dados e as técnicas de modelagem de dados que permitam ao aluno o desenvolvimento de sistemas computacionais com acesso a banco de dados, através de uma linguagem de programação de alto nível.


- Ao final do curso o aluno deverá ser capaz de:
- Avaliar os sistemas de Banco de dados existentes;
- Elaborar modelo conceitual através da abordagem entidade-relacionamento;
- Criar fisicamente um banco de dados;
- Implementar rotinas de programação para conexão e manipulação de um banco de dados em Java.
- Criar um software com acesso ao SGDB MYSQL.

Carga horária

90 horas – 6 créditos.

Agenda de atividades/ Cronograma

- Verifique com atenção o EVA, organize-se para acessar periodicamente o espaço da Disciplina. O sucesso nos seus estudos depende da priorização do tempo para a leitura; da realização de análises e sínteses do conteúdo; e da interação com os seus colegas e tutor.
- Não perca os prazos das atividades. Registre no espaço a seguir as datas, com base no cronograma da disciplina disponibilizado no EVA.
- Use o quadro para agendar e programar as atividades relativas ao desenvolvimento da Disciplina.

Atividades obrigatórias	
Demais atividades (registro pessoal)	

UNIDADE 1

1

Sistema computacional como solução do problema



Objetivos de aprendizagem

- Compreender o papel do desenvolvedor de *softwares*.
- Entender o conceito de problema relacionado à computação.
- Compreender o ciclo de vida de um *software*.
- Analisar e transformar um problema em solução computacional.



Seções de estudo

- Seção 1** Analisando o problema.
- Seção 2** Apresentando o problema.
- Seção 3** Criando os relacionamentos entre os dados.
- Seção 4** Modelo de dados do sistema.
- Seção 5** Criação do *script* SQL.



Para início de conversa

Uma das grandes contribuições da informática para a sociedade atual é a generalização das qualidades profissionais de cada indivíduo. Em outras palavras significa dizer que o famoso “jeitinho brasileiro” está se transformando em um atributo muito desejado nos profissionais atuais, independente da sua área de atuação.

Essa generalização permite que as pessoas realizem trabalhos de outros profissionais de uma maneira aceitável, com qualidade, a partir da sua capacidade de adquirir novas aptidões.

Se você voltar um pouco no tempo, poderá se recordar de algumas profissões que eram essenciais ao dia-a-dia de uma grande empresa. As empresas careciam, por exemplo, de uma exímia secretária, com grandes qualidades de datilógrafa, qualidade essa, decisiva na sua contratação.

Um outro exemplo seria o arquivista responsável por coordenar todos os processos de armazenamento de documentos importantes da empresa, catalogando-os e colocando-os em ordem naqueles enormes fichários, alguns deles, até mesmo, com segredos, como se fossem cofres com um grande tesouro guardado a sete chaves.

Eu poderia citar aqui várias outras profissões que foram evoluindo, fundindo-se a outras que surgiram nos tempos atuais, principalmente em um país em desenvolvimento como o nosso, no qual qualidades profissionais agregadas podem fazer muita diferença na hora da contratação de um profissional. Além de significar uma economia ao contratante.

Há os que questionam essa transformação, avaliando-a como uma forma de exclusão social, justificando que aqueles que possuem capacidade e condições financeiras para se adequar às mais diversas habilidades profissionais têm maior chance de sucesso profissional no mundo globalizado.

Eu considero essa transformação como uma evolução e uma necessidade.

Na área de desenvolvimento de *softwares* essa mesma transformação vem ocorrendo. As habilidades de um profissional envolvido com análise, armazenamento e implementação de sistemas e banco de dados vêm se fundindo para um profissional que abrace as três tecnologias.

O profissional de desenvolvimento de *softwares* precisa ter a habilidade para identificar as reais necessidades dos seus clientes e a capacidade técnica para transformar tudo isso em uma solução computacional.

Que vale salientar, não é tarefa das mais fáceis.

Agregue a isso a necessidade de garantir que os dados armazenados pelo sistema representem fielmente a realidade do dia-a-dia de seus usuários, com integridade e facilidade de acesso.

Essa capacidade de se adaptar às mais diversas necessidades é alcançada a partir da experiência profissional adquirida no dia-a-dia, adicionada de uma dose de boa vontade e estudo, muito estudo, afinal, essa área requer uma constante reciclagem.

Dessa forma, nesta unidade você é convidando a um contato com a área da computação que envolve análise, implementação e armazenamento de dados, de forma prática, fazendo com que você tenha uma visão mais aprofundada do papel do profissional de banco de dados como gerador de soluções computacionais, e mais do que isso, que você seja peça fundamental no processo de criação de um sistema computacional com acesso a banco de dados.

Portanto, mais do que nunca, mãos à obra!

SEÇÃO 1 – Analisando o problema

Se você fizer uma busca em alguns dicionários da língua portuguesa pela palavra “problema”, encontrará algo como:



Problema: questão que se propõe para ser resolvida; algo difícil de explicar; dúvida; questão; mistério; enigma.

Até aí, acredito que não seja nenhuma novidade essa definição de problema, certo? Porém, qual a relação entre a palavra problema e as tarefas executadas por um profissional de informática, em especial aqueles que possuem como principal objetivo a implementação de *softwares*?

Pois bem, o profissional de informática envolvido com o desenvolvimento de sistemas computacionais, ou simplesmente *softwares*, precisa realizar uma tarefa muito complexa, que é compreender as necessidades do cliente a partir do **problema** apresentado. Esse profissional é responsável pelo desenvolvimento e manutenção de sistemas de informação para qualquer tipo de organização, seja ela pública ou privada, de grande porte ou não.

A análise detalhada do problema é fundamental para se levantar os requisitos necessários para que o *software* a ser implementado atenda realmente as necessidades dos seus usuários, ou, em outras palavras, solucione os seus problemas.

A análise do problema tem como principais finalidades:

- o levantamento de requisitos;
- a modelagem e especificação das funcionalidades e delimitações do sistema;
- a definição dos módulos do sistema.

A partir da análise das necessidades do cliente é possível:

- definir em qual plataforma (sistema operacional, ferramenta de banco de dados e linguagem de programação) o sistema será desenvolvido;

- apresentar um Projeto Lógico e um Projeto Físico para implementação e funcionamento do sistema;
- prever o período de testes e treinamento dos usuários e, quando necessário, indicar as melhores soluções para a manutenção do sistema.

Vale uma observação importante de que existe uma diferença muito clara entre **projeto lógico** e **projeto físico**, conforme é mostrado na tabela a seguir:

QUADRO 1.1 - COMPARATIVO ENTRE O PROJETO LÓGICO E PROJETO FÍSICO

Projeto Lógico	Projeto Físico
O objetivo do projeto lógico é a especificação detalhada dos elementos do <i>software</i> quanto à lógica, sem se preocupar com as ferramentas que serão utilizadas.	Tendo como base o Projeto Lógico, o objetivo do projeto físico é detalhar os elementos do <i>software</i> quanto ao físico, se preocupando com as ferramentas computacionais que serão utilizadas.
Finalidade: captação das informações, análise e correções;	Finalidade: especificação técnica completa do <i>software</i> , visando a sua implementação, como linguagem de programação e o sistema de banco de dados utilizado.

Quando se trabalha com desenvolvimento de *softwares*, duas palavras são muito comuns: **implementação** e **implantação**, porém, apesar de sonoridades parecidas, cada uma delas possui uma finalidade única no processo de criação de um *software*. Veja as definições a seguir.



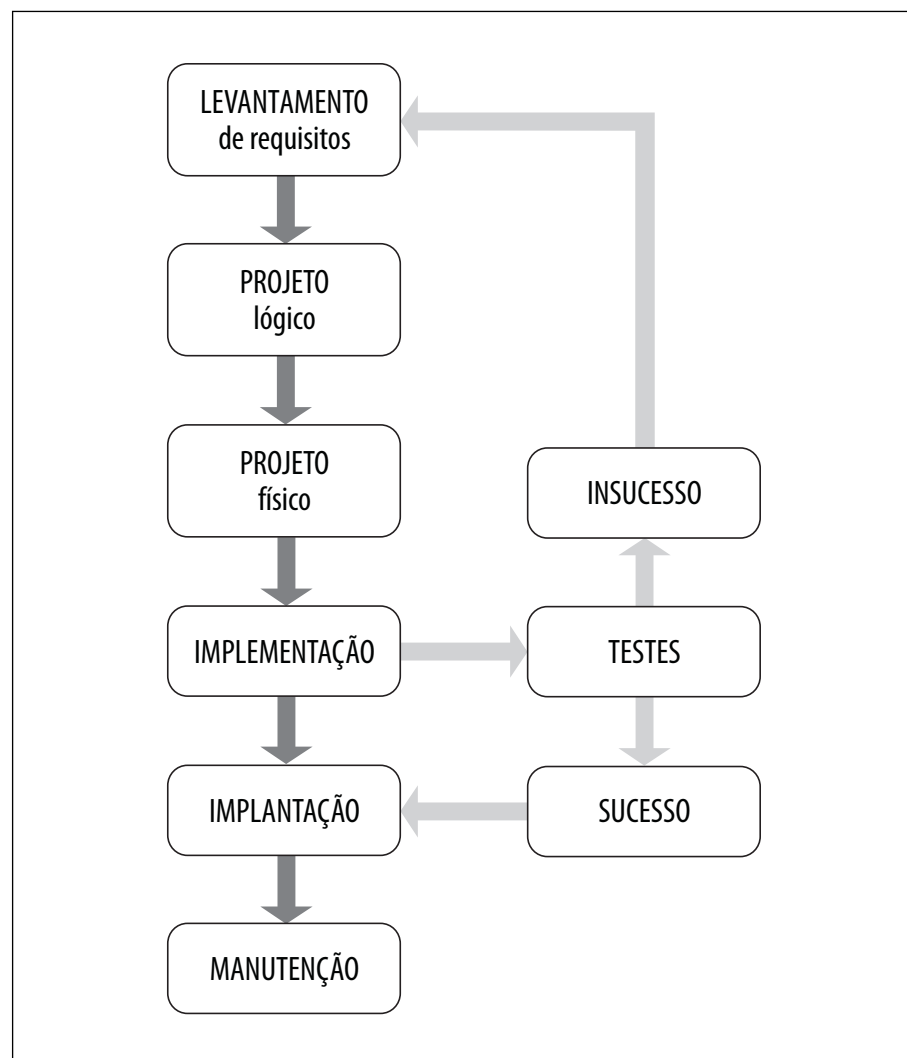
Implementação: tarefa de desenvolvimento e simulação do *software* especificado nos projetos lógico e físico. Tem como resultado os programas de código fonte, as modelagens do sistema e do banco de dados, bem como as documentações de regras de funcionamento do sistema desenvolvido.

Implantação: tem como objetivo o treinamento do usuário, as configurações e a instalação do sistema implementado no cliente, como um produto final, constando de manual do usuário e termos de garantia.

Note que cada uma destas tarefas - definição do Projeto Lógico, Projeto Físico, implementação e implantação - possui uma ordem cronológica. Não é possível implantar um sistema sem que ele tenha sido implementado. Por outro lado, fica complicado implementá-lo sem que sejam definidos os projetos Lógico e Físico.

Essa **ordem cronológica** recebe o nome de **ciclo de vida** do sistema computacional, e pode ser representada pelo seguinte organograma:

GRÁFICO 1.1 - CICLO DE VIDA DO SOFTWARE



Como em qualquer área de desenvolvimento, a etapa de **planejamento** é fundamental. No organograma apresentado, pode-se ver que os processos de levantamento de requisitos, projetos Lógico e Físico, e a implementação podem acontecer repetidas e inúmeras vezes até que o teste do sistema chegue ao resultado esperado pelo usuário-final.



Quanto maior a especificação dos requisitos funcionais do sistema, com um alto nível de análise e avaliação dos dados coletados, maior a probabilidade de se alcançar o sucesso desejado na etapa de testes do sistema.

Lembre-se que quanto mais tempo levar para chegar ao resultado desejado, mais caro se torna o projeto - além de gerar no cliente que contratou os serviços certa desconfiança sobre a qualidade do produto final.

Para achar o ponto ideal na administração do tempo e da qualidade, é necessária a experiência do dia-a-dia. Cada projeto de implementação de *software* possui uma particularidade, na qual, mesmo semelhante a outros projetos já desenvolvidos, pode apresentar funcionalidades e necessidades diferentes.

A experiência de cada novo projeto é que vai lapidando a capacidade do desenvolvedor de *software* em analisar requisitos e transformá-los em *softwares*.

— *Por isso, não se assuste, pois esse amadurecimento leva tempo, requer paciência, dedicação e estudo!*

Na próxima seção você verá o problema proposto por mim para ser analisado e transformado em um sistema computacional.

Quem sabe, não começa aí a sua grande carreira como desenvolvedor de *softwares*?

Até lá!

SEÇÃO 2 – Apresentando o problema

Até aqui você já teve um contato inicial com a tarefa de levantamento de requisitos de um sistema. A partir desta seção, você terá o primeiro contato com o problema sugerido para ser analisado, especificado e, principalmente, solucionado na forma de um sistema computacional, ou simplesmente, um *software*.

Acompanhe com atenção esse processo!

O problema

O proprietário da escola “Sabe Tudo”, deseja implantar na sua escola um sistema para controle acadêmico de seus alunos. Apesar de a escola ser antiga, bem conceituada e possuir vários alunos, não há no setor administrativo nenhum sistema computacional para cadastramento dos seus alunos.

Todos os cadastros de alunos, disciplinas e matrículas são realizados em uma planilha de computador, sem controle algum de acesso. Além de tornar o processo muito demorado, vários problemas ocorrem, como:

- os mesmos dados são cadastrados várias vezes;
- os registros são cadastrados de forma incompleta;
- os alunos são cadastrados em turmas erradas;
- não há controle de alunos matriculados, nem de turmas com vagas;
- faltam dados nos registros;
- as pesquisas por dados de alunos, cursos, turmas e disciplinas são muito demoradas e muitas vezes os pais dos alunos reclamam da demora no atendimento.

Dessa forma, ele busca implantar um sistema que permita o cadastro dos alunos, professores, matrículas e disciplinas. Esse sistema deve rodar em um servidor que ficará instalado na sala da diretoria e todos os funcionários terão acesso ao sistema por meio de computadores instalados no apoio pedagógico e na sala dos professores.

Um ponto importante relatado pelo dono da escola é que o sistema deve ter um custo acessível, pois não é intenção dele um alto investimento na área de informática no momento.

Levantamento dos dados

Com base no problema relatado, podem-se fazer alguns levantamentos importantes, como:

- o sistema deverá ter um controle de acesso de usuários;
- o sistema deverá permitir o cadastramento de dados dos alunos, professores, disciplinas e realizar a matrícula dos alunos;
- o sistema deverá funcionar em uma plataforma cliente-servidor, na qual o banco de dados estará instalado em um servidor na sala da diretoria e as máquinas cliente serão acessadas de dois pontos distintos, como a sala de apoio e sala dos professores;
- alguns dados devem ser obrigatoriamente cadastrados;
- dados repetidos não podem ser cadastrados;
- o sistema deve possibilitar formas de pesquisas aos alunos, cursos, disciplinas e turma com vagas;
- o sistema deve evitar o cadastramento de alunos em turmas erradas.

Dados complementares

Como os dados atuais estão defasados, o dono da escola apresentou as fichas cadastrais de alunos, professores e matrículas que devem ser usadas pelo sistema computacional. A seguir são apresentadas as referidas fichas cadastrais:

TABELA 1.1 - FICHA CADASTRAL DOS ALUNOS


Sabe Tudo 			
Ficha Cadastral - ALUNOS			Nº 15789
Nome:			
Data de Nascimento:		Idade: anos.	
CPF:	Carteira de Identidade:		
Nome do Pai:			
Nome da Mãe:			
Endereço:			Número:
Cidade:	Estado:		
Bairro:			
Telefone:		Celular:	
Em caso de acidentes chamar por:			
Alérgico a:			
Grupo Sangüíneo:			
Doador :	Sim	Não	

TABELA 1.2 - FICHA CADASTRAL DOS PROFESSORES



Sabe Tudo 			
Ficha Cadastral - Professor			Código: 1234.
Nome:			
Data de Nascimento:		Idade: anos.	
CPF:	Carteira de Identidade:		
Carteira de Trabalho Nº:			
Graduado em:			
Estado Civil:			Filhos:
Endereço:			Número:
Cidade:	Estado:		
Bairro:			
Telefone:		Celular:	
Grupo Sangüíneo:			
Doador :	Sim	Não	

TABELA 1.3 - FICHA DE MATRÍCULA

Sabe Tudo 	
Ficha Cadastral - Matrícula	
Aluno:	
Turma:	
Data:	
Valor:	

Com base nessas fichas cadastrais já se pode ter uma idéia dos dados que serão cadastrados no sistema, porém é necessário que se converta esses dados coletados para forma de um **modelo de entidade relacional - MER**.

Pelo modelo de entidade relacional será possível estipular os campos obrigatórios, chaves primárias e estrangeiras do banco de dados que deverá ser implementado.

SEÇÃO 3 – Criando os relacionamentos entre os dados

Nesse primeiro passo, a modelagem de dados representará os conjuntos de dados apresentados pelas fichas cadastrais, com os seus respectivos relacionamentos.



Lembre-se que o que define o relacionamento entre tabelas de um banco de dados é a necessidade de o dado de uma determinada tabela existir obrigatoriamente em outra tabela. Dessa forma, evita-se a redundância dos dados cadastrados.

Em uma visão bem macro, podemos modelar o problema como sendo a integração de 3 tabelas: Alunos, Matrículas e Turmas:

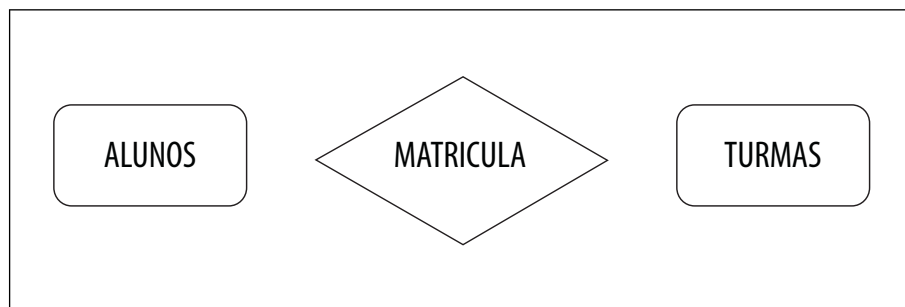


FIGURA 1.1 - MODELO DE DADOS – VISÃO GERAL

No modelo inicial já se destaca uma propriedade importante, pois para que seja feita uma matrícula é necessário que o aluno esteja cadastrado na Tabela de Alunos, e que a turma escolhida esteja cadastrada na Tabela de Turmas.

Dessa forma, a Tabela de Matrículas é uma tabela do tipo associativa, que interliga as tabelas Alunos e Turmas.

Veja a seguir o mesmo modelo já com as cardinalidades entre as tabelas:

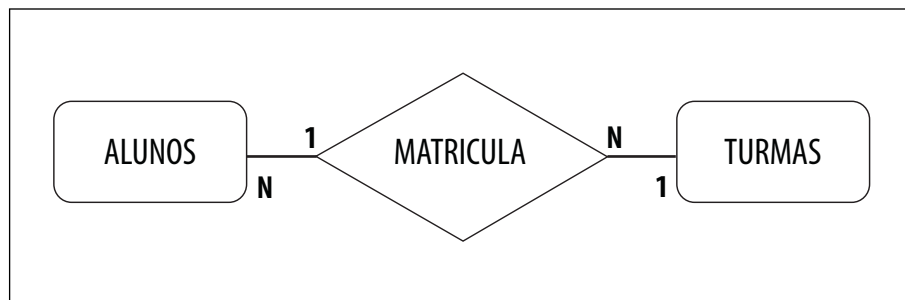


FIGURA 1.2 - MODELO DE DADOS – VISÃO GERAL COM AS CARDINALIDADES

De acordo com a última figura, tem-se as seguintes regras:

- um aluno só pode fazer matrícula em uma única turma;
- uma turma, por sua vez, pode conter N alunos.

Com base nos dados levantados, pode-se acrescentar mais uma tabela ao modelo anterior, que é a Tabela de Professores:

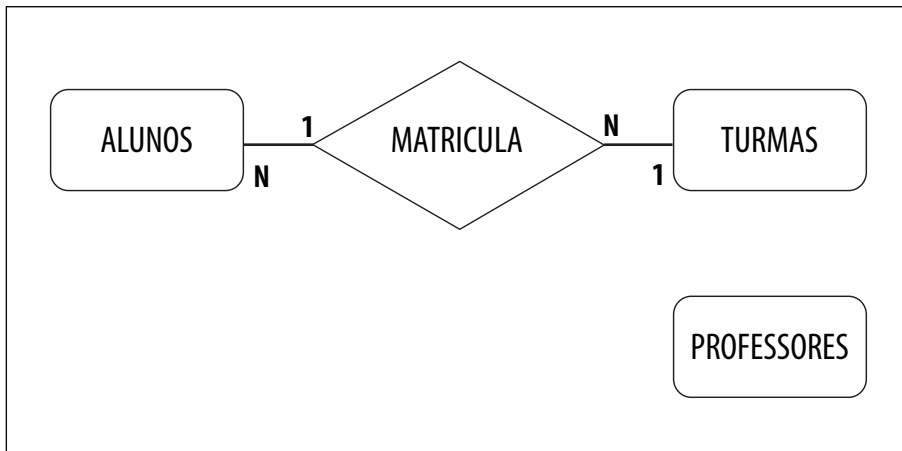


FIGURA 1.3 - MODELO DE DADOS – VISÃO GERAL COM AS CARDINALIDADES E A TABELA DE PROFESSORES

Note que a Tabela de Professores não se relaciona com alunos e nem com matrículas, porém, essa tabela de professores deve se relacionar com a Tabela de Turmas, pois dessa forma sabe-se quais os professores de determinada turma de aula. O modelo de dados ficará da seguinte maneira:

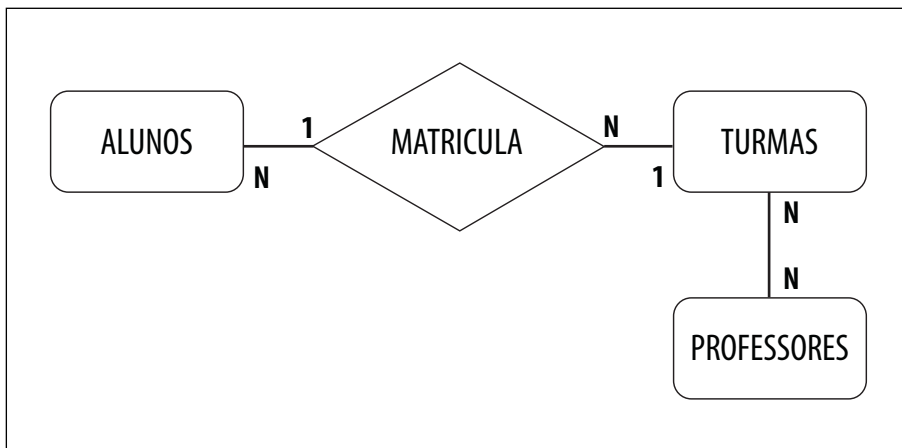


FIGURA 1.4 - MODELO DE DADOS – VISÃO GERAL COM A CARDINALIDADES N PARA N

Há um problema referente à modelagem de um banco de dados: não se pode ter um relacionamento N para N entre tabelas. Para solucionar tal problema, será colocada uma tabela associativa

entre Turmas e Professores, para registrar em qual turma cada professor está locado. Veja como ficará o modelo de dados:

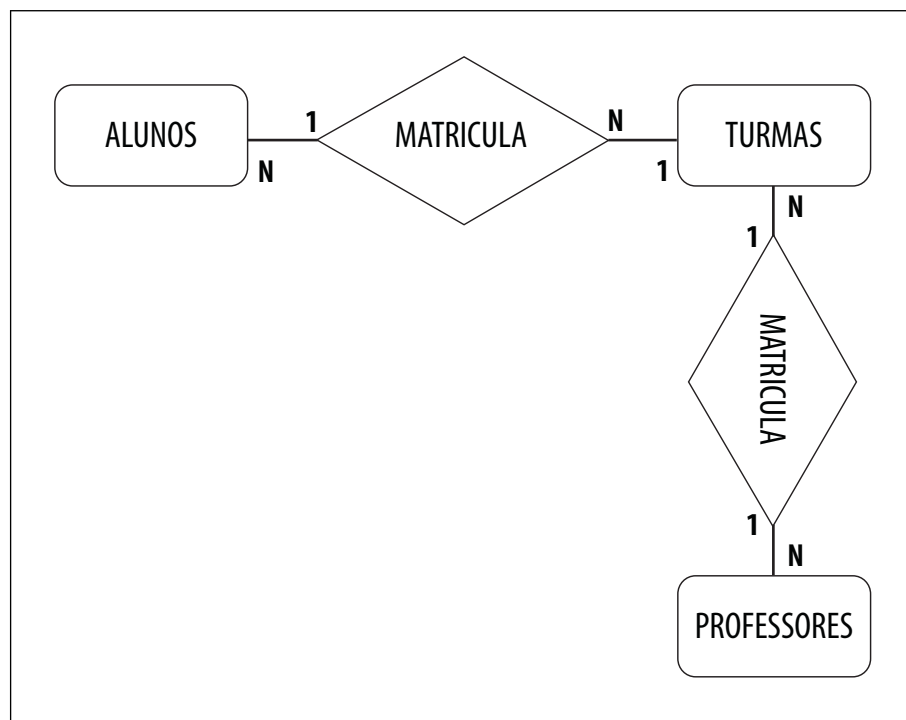


FIGURA 1.5 - MODELO DE DADOS – VISÃO GERAL COM A TABELA ASSOCIATIVA ALOCAÇÕES

Com isso está encerrado o processo de modelagem de dados.

Nas próximas seções será necessário especificar os campos de cada tabela, criar um *script* em linguagem SQL que implemente o modelo desenhado com os tipos de dado de cada campo e suas regras de manipulação.

Então, até lá.

SEÇÃO 4 – Modelo de dados do sistema

Na seção anterior você viu um modelo de dados bem geral, que especificava apenas as tabelas do modelo, sem levar em consideração os campos e os seus respectivos tipos de dados.

A partir de agora, com base nos dados cadastrados em cada ficha utilizada na escola “Sabe Tudo”, você criará um modelo de dados completo, que será responsável pelo armazenamento de todas as informações inseridas e manipuladas pelo sistema.

Tabela de Alunos

A Tabela de Alunos representa fielmente os dados da sua respectiva ficha cadastral.

TABELA 1.4 - TABELA DE ALUNOS COM OS CAMPOS E TIPOS DE DADOS

alunos	
CAMPO	TIPO DE CAMPO
Matricula *	int
Nome	varchar(40)
Data_Nascimento	date
Idade	int
CPF	varchar(14)
RG_Identidade	varchar(20)
Nome_Pai	varchar(40)
Nome_Mae	varchar(40)
Endereco	varchar(50)
Numero	int
Cidade	varchar(35)
Estado	char(2)
Bairro	varchar(40)
Telefone	varchar(30)
Celular	varchar(30)
ChamarPor	varchar(30)
Alergia	varchar(30)
Grupo_Sangue	varchar(10)
Doador	int

Tabela de Professores

A Tabela de Professores representa fielmente os dados da sua ficha cadastral.

TABELA 1.5 - TABELA DE PROFESSORES COM OS CAMPOS E TIPOS DE DADOS



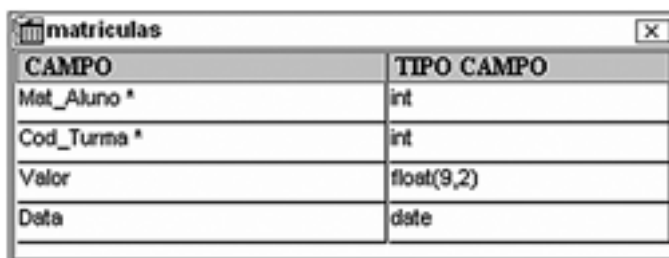
CAMPO	TIPO DE CAMPO
Codigo *	int
Nome	varchar(40)
Data_Nascimento	date
Idade	int
CPF	varchar(14)
RG_Identidade	varchar(20)
CTPS	varchar(20)
Graduacao	varchar(30)
Estado_Civil	varchar(25)
Filhos	int
Endereco	varchar(50)
Numero	int
Cidade	varchar(35)
Bairro	varchar(30)
Estado	char(2)
Telefone	varchar(25)
Celular	varchar(25)
Grupo_Sangue	varchar(10)
Doador	int

Tabela de Matrículas

A Tabela de Matrículas representa a relação entre o aluno cadastrado na Tabela de Alunos e a turma cadastrada na Tabela de Turmas. Como os dados serão armazenados em um banco de dados, será utilizado o relacionamento entre as tabelas, por meio das chaves estrangeiras, a fim de garantir a integridade dos dados cadastrados.

Dessa forma a Tabela de Matrículas ficará desta forma:

TABELA 1.6 - TABELA DE MATRÍCULAS COM OS CAMPOS E TIPOS DE DADOS

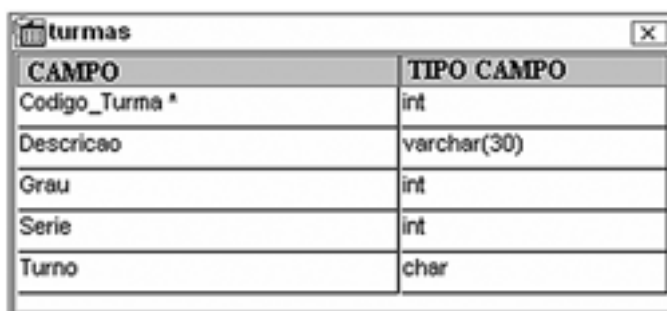


CAMPO	TIPO CAMPO
Mat_Aluno *	int
Cod_Turma *	int
Valor	float(9,2)
Data	date

Tabela de Turmas

A Tabela de Turmas não possui uma ficha cadastral padrão, porém com a análise do problema pode-se chegar à seguinte estrutura:

TABELA 1.7 - TABELA DE TURMAS COM OS CAMPOS E TIPOS DE CAMPOS

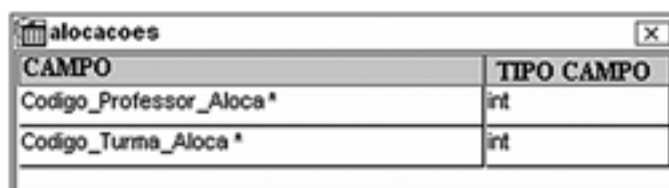


CAMPO	TIPO CAMPO
Codigo_Turma *	int
Descricao	varchar(30)
Grau	int
Serie	int
Turno	char

Tabela de Alocações

A Tabela de Alocações também não possui uma ficha padrão, porém a sua finalidade é bem clara, essa tabela representa a relação entre o professor e a turma para qual ele deverá lecionar. A sua estrutura ficará assim:

TABELA 1.8 - TABELA DE ALOCAÇÕES COM OS CAMPOS E TIPOS DE CAMPOS



CAMPO	TIPO CAMPO
Codigo_Professor_Aloca *	int
Codigo_Turma_Aloca *	int

Com isso, o modelo de dados está composto por todas as tabelas, com seus respectivos campos e tipos de campos. Veja a seguir o modelo de dados completo:

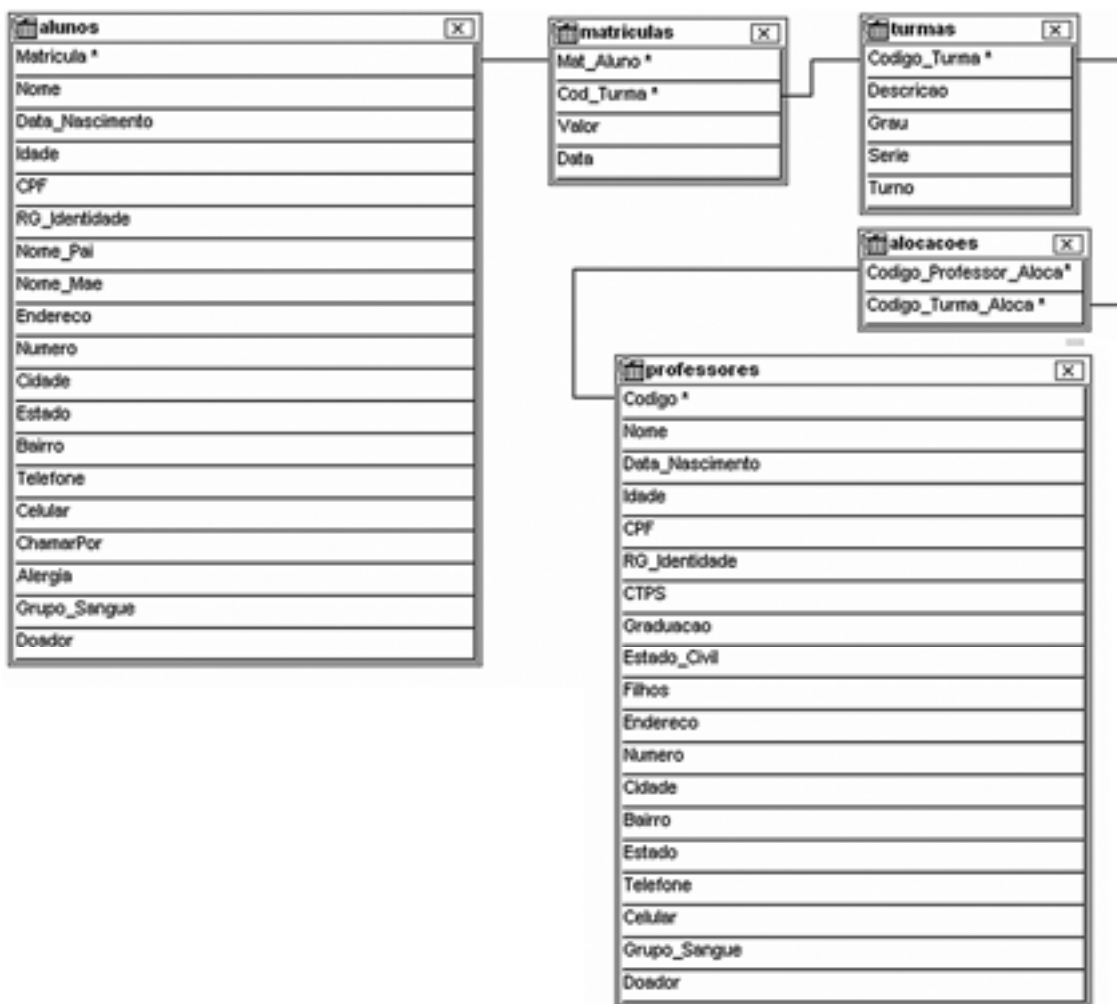


FIGURA 1.6 - MODELO DE DADOS COMPLETO

Esse modelo é uma representação gráfica, para visualização e interpretação das regras do banco de dados que será criado.

O próximo passo é converter esse modelo para um conjunto de comandos de definição de dados da linguagem SQL (*Structure Query Language*), chamados de **script SQL**.

SEÇÃO 5 – Criação do **script SQL**

O **script SQL** é um arquivo escrito em qualquer tipo de editor de texto, como por exemplo, o Bloco de Notas do Windows. O ideal é salvar esse arquivo com a extensão **.sql** para indicar que se trata de um **script SQL**.

Nesse arquivo são escritos os comandos em SQL que realizam as seguintes tarefas:

- criação das tabelas;
- definição das colunas de cada tabela (campos);
- definição dos tipos de dados de cada coluna (campos);
- definição das regras de chave primária e chave estrangeira de cada tabela.

O **script** em SQL de criação do banco de dados está diretamente relacionada aos comandos de definição de dados da linguagem SQL, que são: *Create Table*, *Drop Table* e *Alter Table*.

Para facilitar, serão apresentados os **scripts** de cada uma das tabelas do modelo de dados da seção anterior separadamente, e por último o **script** por completo.

Criação da Tabela de Alunos

Veja a seguir o *script* de criação da Tabela de Alunos:

```
CREATE TABLE ALUNOS (
  Matricula          int(11) NOT NULL,
  Nome               varchar(40) NOT NULL,
  Data_Nascimento    Date NOT NULL,
  Idade              int(4) NOT NULL,
  CPF                varchar(14),
  RG_Identidade      varchar(20),
  Nome_Pai           varchar(40) NOT NULL,
  Nome_Mae           varchar(40) NOT NULL,
  Endereco           varchar(50) NOT NULL,
  Numero             int(4) NOT NULL,
  Cidade             varchar(35) NOT NULL,
  Estado             char(2) NOT NULL,
  Bairro             varchar(40) NOT NULL,
  Telefone           varchar(30),
  Celular            varchar(30),
  ChamarPor          varchar(30),
  Alergia            varchar(30),
  Grupo_Sangue       varchar(10),
  Doador             int(4),
  PRIMARY KEY        ( Matricula )
);
```

Com base nesse *script*, a estrutura da tabela de alunos possui as seguintes regras:

TABELA 1.9 - REGRAS DA TABELA DE ALUNOS

CHAVE PRIMARIA	CAMPO	TIPO	TAM.	DECIMAIS	OBRIGATÓRIO
✓	Matricula	INTEGER	11	0	✓
	Nome	VARCHAR	40	0	✓
	Data_Nascimento	DATE	10	0	✓
	Idade	INTEGER	4	0	✓
	CPF	VARCHAR	14	0	
	RG_Identidade	VARCHAR	20	0	
	Nome_Pai	VARCHAR	40	0	✓
	Nome_Mae	VARCHAR	40	0	✓
	Endereco	VARCHAR	50	0	✓
	Numero	INTEGER	4	0	✓
	Cidade	VARCHAR	35	0	✓
	Estado	CHAR	2	0	✓
	Bairro	VARCHAR	40	0	✓
	Telefone	VARCHAR	30	0	
	Celular	VARCHAR	30	0	
	ChamarPor	VARCHAR	30	0	
	Alergia	VARCHAR	30	0	
	Grupo_Sangue	VARCHAR	10	0	
	Doador	INTEGER	4	0	

A tabela anterior representa o tipo de cada coluna, o seu nome, se a mesma é obrigatória e se é chave primária.

Criação da Tabela de Professores

```
CREATE TABLE PROFESSORES (
    Codigo                int(4) NOT NULL,
    Nome                  varchar(40) NOT NULL,
    Data_Nascimento       Date NOT NULL ,
    Idade                 int(4) NOT NULL,
    CPF                   varchar(14) NOT NULL ,
    RG_Identidade         varchar(20) NOT NULL,
    CTPS                  varchar(20) NOT NULL,
    Graduacao            varchar(30) NOT NULL,
    Estado_Civil          varchar(25) NOT NULL,
    Filhos                int(4) NOT NULL,
    Endereco              varchar(50) NOT NULL ,
    Numero                int(4) NOT NULL,
    Cidade                varchar(35) NOT NULL,
    Bairro                varchar(30) NOT NULL ,
    Estado                char(2) NOT NULL,
    Telefone              varchar(25) NOT NULL,
    Celular               varchar(25),
    Grupo_Sangue          varchar(10),
    Doador                int(4),
    PRIMARY KEY           ( Codigo )
);
```

Com base neste *script*, a estrutura da Tabela de Professores possui as seguintes regras:

TABELA 1.10 - REGRAS DA TABELA DE PROFESSORES

CHAVE PRIMÁRIA	CAMPO	TIPO	TAM.	DECIMAIS	OBRIGATÓRIO
✓	Codigo	INTEGER	4	0	✓
	Nome	VARCHAR	40	0	✓
	Data_Nascimento	DATE	10	0	✓
	Idade	INTEGER	4	0	✓
	CPF	VARCHAR	14	0	✓
	RG_Identidade	VARCHAR	20	0	✓
	CTPS	VARCHAR	20	0	✓
	Graduacao	VARCHAR	30	0	✓
	Estado_Civil	VARCHAR	25	0	✓
	Filhos	INTEGER	4	0	✓
	Endereco	VARCHAR	50	0	✓
	Numero	INTEGER	4	0	✓
	Cidade	VARCHAR	35	0	✓
	Bairro	VARCHAR	30	0	✓
	Estado	CHAR	2	0	✓
	Telefone	VARCHAR	25	0	✓
	Celular	VARCHAR	25	0	
	Grupo_Sangue	VARCHAR	10	0	
	Doador	INTEGER	4	0	

Criação da Tabela de Turmas

```
CREATE TABLE TURMAS (
  Codigo_Turma    int(4) NOT NULL,
  Descricao       varchar(30) NOT NULL,
  Grau            int(4) NOT NULL,
  Serie           int(4) NOT NULL,
  Turno           char(1) NOT NULL,
  PRIMARY KEY     ( Codigo_Turma )
);
```

Com base nesse *script*, a estrutura da Tabela de Turmas possui as seguintes regras:

TABELA 1.11 - REGRAS DA TABELA DE TURMAS

CHAVE PRIMÁRIA	CAMPO	TIPO	TAM.	DECIMAIS	OBRIGATÓRIO
✓	Codigo_Turma	INTEGER	4	0	✓
	Descricao	VARCHAR	30	0	✓
	Grau	INTEGER	4	0	✓
	Serie	INTEGER	4	0	✓
	Turno	CHAR	1	0	✓

Para finalizar, a criação das tabelas associativas: Matrículas e Alocações.

Criação da Tabela de Matrículas

```
CREATE TABLE MATRICULAS (
  Mat_Aluno       int(11) NOT NULL,
  Cod_Turma       int(4) NOT NULL,
  Valor           float (9,2) NOT NULL,
  Data Date       NOT NULL,
  PRIMARY KEY     (Mat_Aluno, Cod_Turma),
  Foreign Key (Mat_Aluno) References Alunos (Matricula),
  Foreign Key (Cod_Turma) References Turmas (Codigo_Turma)
);
```

Com base neste *script*, a estrutura da Tabela de Matrículas possui as seguintes regras:

TABELA 1.12 - REGRAS DA TABELA DE MATRÍCULAS

CHAVE PRIMÁRIA	CHAVE ESTRANGEIRA	CAMPO	TIPO	TAM.	DECIMAIS	OBRIGATÓRIO
✓	✓	Mat_Aluno	INTEGER	4	0	✓
✓	✓	Cod_Turma	INTEGER	4	0	✓
		Valor	FLOAT	9	2	✓
		Data	DATE	10	0	✓

Criação da Tabela de Alocações

```
CREATE TABLE ALOCACOES (
  Codigo_Professor_Aloca          int(4) NOT NULL,
  Codigo_Turma_Aloca              int(4) NOT NULL,
  PRIMARY KEY (Codigo_Professor_Aloca,Codigo_Turma_Aloca ),
  Foreign Key (Codigo_Professor_Aloca) References Professores(Codigo),
  Foreign Key (Codigo_Turma_Aloca)   References Turmas(Codigo_Turma)
);
```

Com base nesse *script*, a estrutura da Tabela de Alocações possui as seguintes regras:

TABELA 1.13 - REGRAS DA TABELA DE ALOCAÇÕES

CHAVE PRIMÁRIA	CHAVE ESTRANGEIRA	CAMPO	TIPO	TAM.	OBRIGATÓRIO
✓	✓	Codigo_Professor_Aloca	INTEGER	4	✓
✓	✓	Codigo_Turma_Aloca	INTEGER	4	✓

O *script* de criação do banco de dados ficou da seguinte forma:

```
-- Script de Criação das Tabelas

-- Remove as Tabelas
DROP TABLE IF EXISTS ALOCACOES;
DROP TABLE IF EXISTS MATRICULAS;
DROP TABLE IF EXISTS ALUNOS;
DROP TABLE IF EXISTS TURMAS;
DROP TABLE IF EXISTS PROFESSORES;
```

SEGUE ►

```
-- Criar a Tabela de Alunos
```

```
CREATE TABLE ALUNOS (
  Matricula          Int(11) NOT NULL,
  Nome               Varchar(40) NOT NULL,
  Data_Nascimento    Date NOT NULL,
  Idade              Int(4) NOT NULL,
  CPF                Varchar(14),
  RG_Identidade      Varchar(20),
  Nome_Pai           Varchar(40) NOT NULL,
  Nome_Mae           Varchar(40) NOT NULL,
  Endereco           Varchar(50) NOT NULL,
  Numero             Int(4) NOT NULL,
  Cidade             Varchar(35) NOT NULL,
  Estado             Char(2) NOT NULL,
  Bairro             Varchar(40) NOT NULL,
  Telefone           Varchar(30),
  Celular            Varchar(30),
  Chamarpor          Varchar(30),
  Alergia            Varchar(30),
  Grupo_Sangue       Varchar(10),
  Doador             Int(4),
  PRIMARY KEY ( Matricula )
);
```

```
-- Criar a Tabela de Professores
```

```
CREATE TABLE PROFESSORES (
  Codigo             Int(4) NOT NULL,
  Nome               Varchar(40) NOT NULL,
  Data_Nascimento    Date NOT NULL ,
  Idade              Int(4) NOT NULL,
  CPF                Varchar(14) NOT NULL ,
  RG_Identidade      Varchar(20) NOT NULL,
  CTPS               Varchar(20) NOT NULL,
  Graduacao          Varchar(30) NOT NULL,
  Estado_Civil       Varchar(25) NOT NULL,
  Filhos             Int(4) NOT NULL,
  Endereco           Varchar(50) NOT NULL ,
  Numero             Int(4) NOT NULL,
  Cidade             Varchar(35) NOT NULL,
  Bairro             Varchar(30) NOT NULL ,
  Estado             Char(2) NOT NULL,
  Telefone           Varchar(25) NOT NULL,
  Celular            Varchar(25),
  Grupo_Sangue       Varchar(10),
  Doador             Int(4),
  PRIMARY KEY ( Codigo )
);
```

```

-- Criar a Tabela de Turmas

CREATE TABLE TURMAS (
  Codigo_Turma      Int(4) NOT NULL,
  Descricao         Varchar(30) NOT NULL,
  Grau             Int(4) NOT NULL,
  Serie            Int(4) NOT NULL,
  Turno            Char(1) NOT NULL,
  PRIMARY KEY ( Codigo_Turma )
);

-- Criar a Tabela de Matrículas

CREATE TABLE MATRICULAS (
  Mat_Aluno         Int(11) NOT NULL,
  Cod_Turma         Int(4) NOT NULL,
  Valor Float(9,2) NOT NULL ,
  Data Date NOT NULL,
  PRIMARY KEY (Mat_Aluno,Cod_Turma),
  FOREIGN KEY(MAT_ALUNO) REFERENCES ALUNOS(MATRICULA),
  FOREIGN KEY(COD_TURMA) REFERENCES TURMAS(CODIGO_TURMA)
);

-- Criar a Tabela de Alocações

CREATE TABLE ALOCACOES (
  Codigo_Professor_Aloca Int(4) NOT NULL ,
  Codigo_Turma_Aloca     Int(4) NOT NULL ,
  PRIMARY KEY ( Codigo_Professor_Aloca,Codigo_Turma_Aloca ),
  FOREIGN KEY (CODIGO_PROFESSOR_ALOCA) REFERENCES PROFESSORES(CODIGO),
  FOREIGN KEY (CODIGO_TURMA_ALOCA) REFERENCES TURMAS(CODIGO_TURMA)
);

```

Bem, com isso você tem a modelagem do banco de dados em um formato que pode ser lido pela ferramenta de banco de dados que será utilizado para implementação do sistema e armazenamento dos dados.

Essa é uma entre várias etapas para se criar um *software*. Na próxima unidade você vai para etapa de preparação, na qual serão especificadas a linguagem de programação, a ferramenta de banco de dados e as funcionalidades do sistema.

Até a próxima unidade!



Síntese

Você pôde estudar nesta unidade que a tarefa de desenvolver *software* requer muita análise por parte do profissional. A principal dificuldade é transformar um problema em uma solução computacional que atenda as necessidades do cliente e seus usuários.

As tarefas de especificação do problema possuem uma cronologia e ordem formal, a inversão dessas tarefas pode resultar em mais tempo de desenvolvimento e conseqüentemente, em maior custo.

Na maioria das vezes, um sistema computacional está agregado a uma solução de banco de dados, responsável por armazenar e garantir a integridade dos dados.

Para se chegar a um modelo de dados ideal e que atenda as necessidades do usuário é importantíssimo um levantamento de dados criterioso e organizado.

A representação do modelo de dados é feita pela modelagem de entidade e relacionamentos, que é uma representação gráfica para maior compreensão da solução que está sendo adotada.

Para transformar esta solução em uma ferramenta de banco de dados, é necessária a criação de um conjunto de comandos em SQL que construa e altere as estruturas modeladas, e esse conjunto de comandos é chamado de *script*.

Um *script* pode mudar de acordo com a ferramenta computacional que se deseja adotar, por algum novo comando que seja necessário, porém a modelagem de entidade e relacionamento é única, independente da ferramenta computacional e é a principal referência para o sistema que será implementado.



Atividades de auto-avaliação

Efetue as atividades de auto-avaliação e, a seguir, acompanhe as respostas e comentários a respeito. Para melhor aproveitamento do seu estudo, realize a conferência de suas respostas somente depois de fazer as atividades propostas.

Leia com atenção os enunciados e realize, a seguir, as atividades.

1. A maioria das profissões exige que o especialista seja formado em um curso de graduação, como médicos e advogados. Na área de desenvolvimento de *softwares* não há exigência de graduação para que um profissional execute as suas atividades. Qual a sua opinião sobre essa questão?

2. Quando se desenvolve um produto, como uma cadeira, uma casa, etc., durante o processo de criação é possível apresentá-lo ao cliente, para seu acompanhamento. Mas, no caso de desenvolvimento de *softwares* não há um produto palpável para ser apresentado. Como agir em uma situação como essa?



Saiba mais

Para aprofundar as questões abordadas nesta unidade, você poderá pesquisar em:

- CHEN, Peter. **Modelagem de dados: a abordagem entidade-relacionamento para projeto lógico**. São Paulo: Makron Books, 1990.
- DATE, C J. **Bancos de dados: fundamentos**. Rio de Janeiro: Campus, 1985.
- _____. **Introdução a sistema de banco de dados**. 8. ed. Rio de Janeiro: Campus, 1990.
- FILHO, Trajano Leme. **Metodologia de desenvolvimento de sistema**. São Paulo: Axcel, 2003.

UNIDADE 2

2

Preparando o ambiente de banco de dados



Objetivos de aprendizagem

- Aprender a instalar e configurar uma ferramenta computacional de banco de dados.
- Executar comandos de criação e manipulação de banco de dados.
- Converter um *script* SQL em um banco de dados.



Seções de estudo

- Seção 1** Critérios para escolha da ferramenta de banco de dados.
- Seção 2** Carregando o MySQL.
- Seção 3** Testando a ferramenta de banco de dados.
- Seção 4** Executando o aplicativo de banco de dados.
- Seção 5** Comandos básicos do MySQL.
- Seção 6** Convertendo o *script* em um banco de dados.



Para início de conversa

Depois de conhecer um pouco sobre o processo de análise e modelagem de um sistema computacional. Nessa unidade você vai aprender a preparar o ambiente no seu computador para o desenvolvimento de um banco de dados.

Até o momento você não precisou se preocupar com isso. O processo de análise representa a preparação para as atividades de implementação e implantação do sistema, sem se preocupar com detalhes de linguagem de programação, banco de dados e equipamentos utilizados.

A partir desta unidade você irá instalar e configurar um sistema de gerenciamento de banco de dados para implementação do modelo de banco de dados criado na unidade anterior, que resultou num *script* em SQL. Para isso, será necessário que você se lembre de alguns comandos em SQL já estudados em outras disciplinas, além de necessitar muita atenção, pois a criação de um banco de dados é cheia de particularidades.

Acredito que você irá gostar muito desta unidade, pois é uma unidade bem prática, na qual você colocará em uso os seus conhecimentos.

Então, vamos praticar!



SEÇÃO 1 – Critérios para a escolha da ferramenta de banco de dados

A seleção da ferramenta de banco de dados é uma tarefa complexa, pois existem várias ferramentas disponíveis no mercado atual, principalmente para os especialistas que não possuem muita experiência nessa área de atuação. Para um leigo no assunto, as ferramentas podem parecer semelhantes, com as mesmas garantias de funcionamento, dificultando ainda mais a escolha.

Independentemente das características do produto, o mais importante é garantir que a ferramenta escolhida atenda as necessidades do cliente e garanta a integridade dos dados. A escolha da ferramenta deve se adequar aos resultados da análise do sistema.

De forma resumida, uma ferramenta de banco de dados deve ter as seguintes características:

- aceitar como linguagem de manipulação e de definição de dados o SQL padrão ANSI, que é internacional;
- permitir a definição de regras de integridade, principalmente de integridade referencial, as chaves estrangeiras;
- possibilitar portabilidade em diferentes plataformas como: Linux e Windows;
- trabalhar na arquitetura cliente-servidor;
- disponibilizar mecanismos que permitam manter o acesso ao banco de dados mesmo em caso de falha no dispositivo de armazenamento.

Cada ferramenta tem sua versão e a cada nova versão novos recursos, que facilitam a criação do banco de dados, são adicionados.

Atualmente, uma característica muito importante tem sido agregada a essas ferramentas, a disponibilidade sem custo algum - as chamadas **ferramentas de código aberto**. Entretanto, tal gratuidade não pode ser a principal razão para se escolher a ferramenta desejada.

Retomemos o projeto da escola “Sabe Tudo”. Para avaliar a ferramenta que será utilizada nesse projeto, um fator importante deverá ser considerado, é a necessidade de se criar um produto de baixo custo. Desta forma, o projeto utilizará uma ferramenta de código aberto, chamada MySQL.

Nas próximas seções, você fará o acesso à ferramenta MySQL pela internet, baixará essa ferramenta e executará o processo de instalação e configuração da mesma.

SEÇÃO 2 – Carregando o MySQL

Para desenvolver o projeto da escola “Saber Tudo”, será usado o sistema de banco de dados chamado **MySQL**, pois seu conjunto de recursos oferecidos é suficiente para o desenvolvimento do projeto.

A ferramenta pode ser acessada em <www.mysql.com/>.

A seguir, são apresentados os passos para baixar o instalador da ferramenta MySQL. Para exemplificar, utilize como base a versão para o sistema operacional Windows.

Baixando o MySQL

1. Acesse o endereço de internet: <www.mysql.com>.
2. Com a página carregada, acesse o link: *Developer Zone*. Este *link* é específico para os desenvolvedores de softwares com acesso a banco de dados;
3. A versão mais atual da ferramenta é MySQL 5.1, porém será utilizada a versão 5.0 por ser uma versão mais estável e de tamanho menor para ser baixado pela internet. Para iniciar o processo clique no *link* **MySQL 5.0**, que está localizado logo no início da página.

Veja a imagem a seguir:

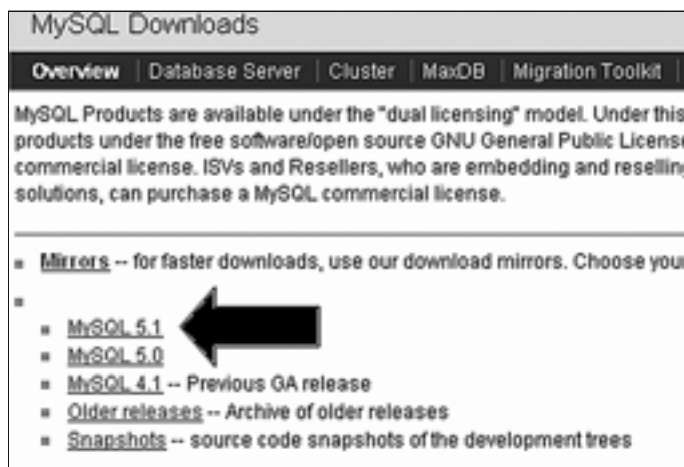


FIGURA 2.1 - LINK PARA DOWNLOAD DA FERRAMENTA DE BANCO DE DADOS

4. Na página que se abre, procure pela versão para o sistema operacional Windows. Localizando o *link* para essa versão, clique na palavra **download** do arquivo *Windows Essentials* para iniciar o processo de *download*. Veja a imagem a seguir:

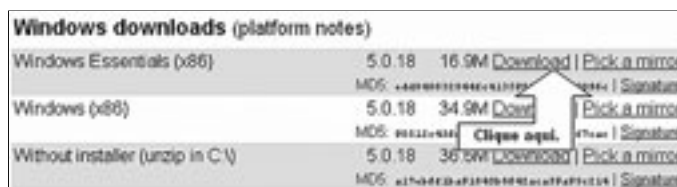


FIGURA 2.2 - LINK PARA DOWNLOAD DA VERSÃO 5.0 PARA WINDOWS

5. Na janela que se abre, clique em **Salvar**:

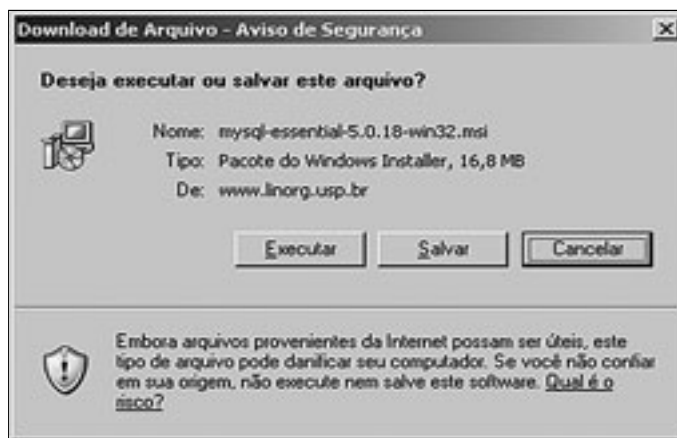


FIGURA 2.3 - INICIANDO O PROCESSO DE DOWNLOAD DA VERSÃO 5.0

6. Especifique a pasta para armazenamento do arquivo e clique em **salvar**:



FIGURA 2.4 - ARMAZENANDO O ARQUIVO DE INSTALAÇÃO NO COMPUTADOR

Agora que o arquivo já foi baixado para sua máquina, a próxima tarefa é instalar o programa.

Passos para fazer a instalação

1. Localize o arquivo armazenado no seu computador.

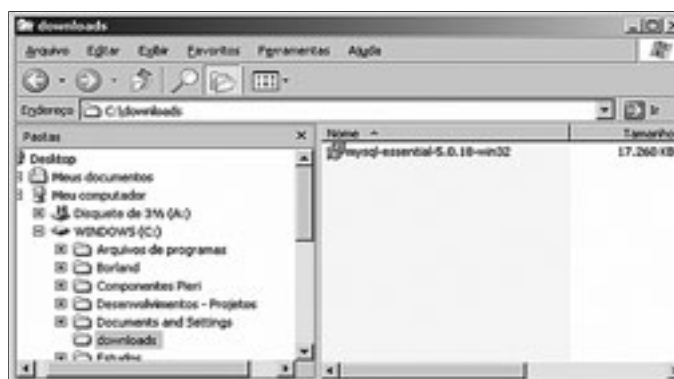


FIGURA 2.5 - ARQUIVO DE INSTALAÇÃO DO MySQL

2. Dê um duplo clique no arquivo mysql-essencial-5.0.18-win32 para iniciar a instalação. Na janela que se abrirá clique no botão **Executar**.

Veja abaixo:

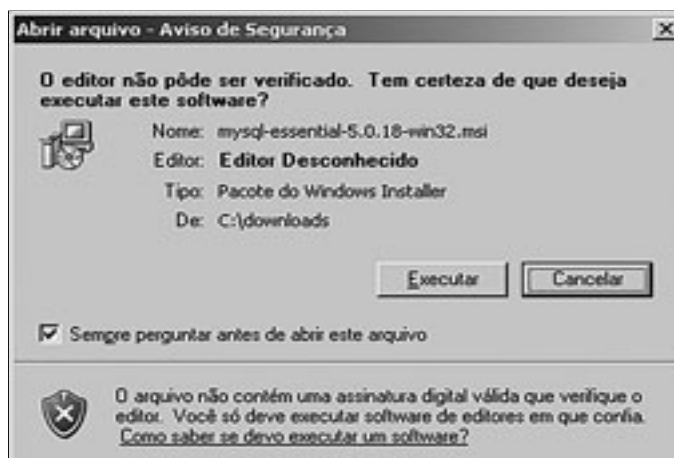


FIGURA 2.6 - INÍCIO DA INSTALAÇÃO DO MySQL 5.0

Com isso, o processo de instalação se iniciará e lhe serão apresentadas algumas telas de configuração.

Passos para cada tela

1. Tela de apresentação – clique no botão *Next*.



FIGURA 2.7 - TELA DE APRESENTAÇÃO

2. Tela de *Setup* – selecione a opção *Typical* e clique no botão *Next*.



FIGURA 2.8 - TELA DE SETUP

3. Tela de início da instalação – clique no botão ***Install***.

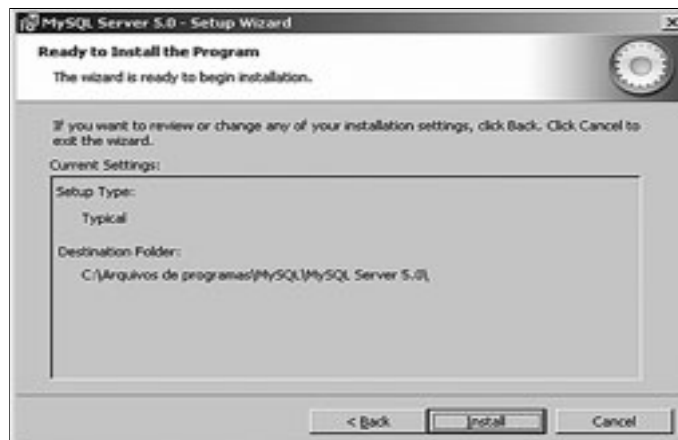


FIGURA 2.9 - TELA DE INÍCIO DA INSTALAÇÃO

4. Tela de assinatura – selecione Skip *Sign-Up* e clique em ***Next***.



FIGURA 2.10 - TELA DE ASSINATURA

5. Tela de encerramento – clique no botão *Finish*.



FIGURA 2.11 - TELA DE ENCERRAMENTO

Bem, com mais essa etapa o banco de dados já está instalado, faltando apenas a configuração do nome de usuário e senha de acesso.

Passos para a configuração

1. Tela de configuração – clique no botão *Next*.



FIGURA 2.12 - TELA DE CONFIGURAÇÃO

2. Tela de servidor – clique no botão *Next*.



FIGURA 2.13 - TELA DE SERVIDOR

3. Tela de tipo de uso – clique no botão *Next*.



FIGURA 2.14 - TELA DE TIPO DE USO

4. Tela de configuração do servidor – clique no botão *Next*.



FIGURA 2.15 - TELA DE CONFIGURAÇÃO DO SERVIDOR

5. Tela de espaço em disco – clique em *Next*.



FIGURA 2.16 - TELA DE ESPAÇO EM DISCO

6. Tela de conexões – clique no botão *Next*.



FIGURA 2.17 - TELA DE CONEXÕES

7. Tela de TCP/IP – clique no botão *Next*.



FIGURA 2.18 - TELA DE TCP/IP

8. Tela de configuração do MySQL como um serviço do Windows – clique no botão *Next*.



FIGURA 2.19 - MYSQL COMO SERVIÇO DO WINDOWS

9. Configuração de senha de acesso – na caixa *New root password* edite a senha **virtual**. Na caixa *Confirm* repita a palavra **virtual**. Essa será a senha de acesso ao MySQL.

Após editar a palavra **virtual**, clique no botão *Next*.



FIGURA 2.20 - TELA DE SEGURANÇA

10. Tela de finalização – clique no botão **Execute** para realizar as configurações e por fim clique em **Finish** para encerrar todo o processo de configuração.



FIGURA 2.21 - TELA DE ENCERRAMENTO DA INSTALAÇÃO E CONFIGURAÇÃO

SEÇÃO 3 – Testando a ferramenta de banco de dados

Agora que a instalação foi realizada, você terá um contato mais direto com essa ferramenta.

Para verificar a instalação realizada anteriormente, clique no botão **Iniciar**, da barra de tarefas do Windows, um novo grupo de programa deverá ser adicionado, conforme mostra a imagem a seguir.

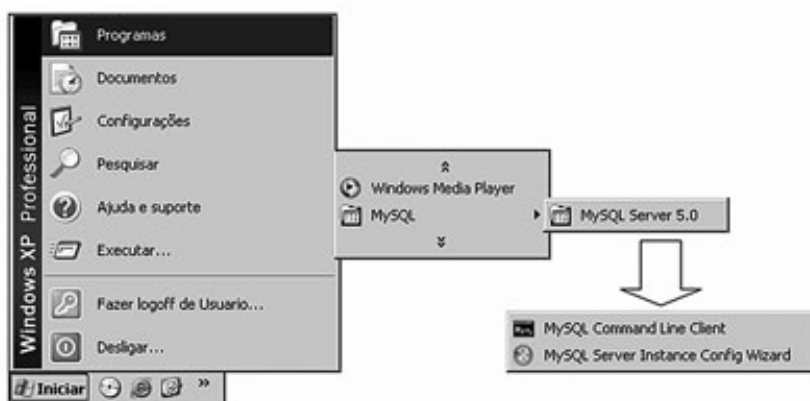


FIGURA 2.22 - GRUPO DE PROGRAMAS DO MYSQL

A sua interação com a ferramenta MySQL será feita por meio do programa *MySQL Command Line Cliente*, que representa o editor de comando em SQL que serão interpretados pelo sistema de gerenciamento do banco de dados.

Para acessar esse aplicativo basta clicar na opção *MySQL Command Line Cliente* do grupo de programas do MySQL.

Veja a tela que aparecerá:



FIGURA 2.23 - TELA DE ACESSO AO MYSQL COMMAND LINE CLIENTE

Nesse caso, é necessário que você informe a senha utilizada na instalação do programa para poder acessar o aplicativo. Para padronizar a instalação, solicitei que fosse informada a senha **virtual**, sendo assim, basta você escrever **virtual** e pressionar a tecla *enter*.

A tela de entrada ao *MySQL Command Line Cliente* pode ser vista a seguir.

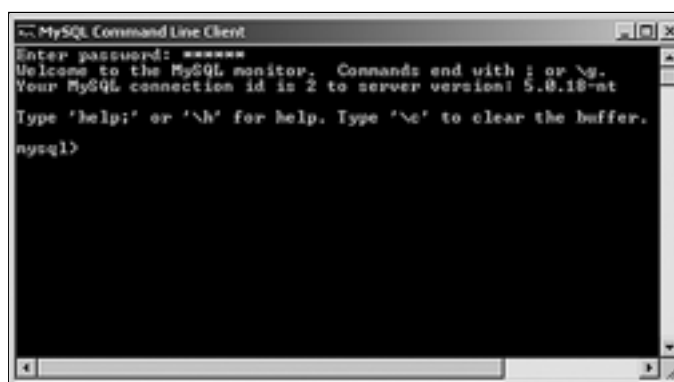


FIGURA 2.24 - EDITOR DE COMANDOS DO MYSQL COMMAND LINE CLIENTE

SEÇÃO 4 – Executando o aplicativo de banco de dados

Bem, agora que você está com tudo pronto, é hora de começar.

Nesta seção você usará alguns comandos básicos da ferramenta MySQL, que são essenciais para a criação e manipulação do banco de dados para o projeto da escola “Sabe Tudo”.

Para facilitar o uso do MySQL, crie um atalho para o aplicativo *MySQL Command Line Client* na área de trabalho.

Caso você não se lembre de como fazer isso, siga o modelo apresentado abaixo:

1. Abra o grupo de programas do MySQL no Windows clicando o botão **Iniciar**.

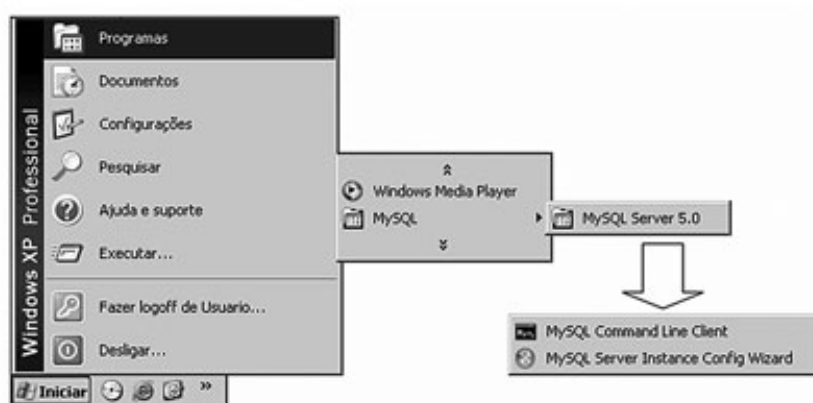


FIGURA 2.25 - GRUPO DE PROGRAMAS DO MYSQL

2. Clique com botão da direita sobre a opção *MySQL Command Line Client* para que apareça o seguinte menu suspenso:

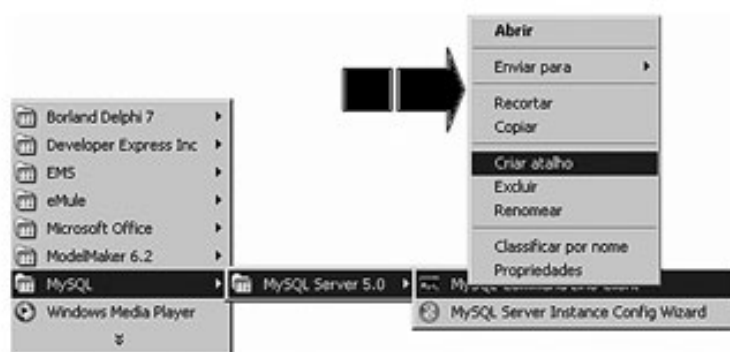


FIGURA 2.26 MENU DE PROPRIEDADES DO WINDOWS

3. Clique na opção **Criar Atalho**:

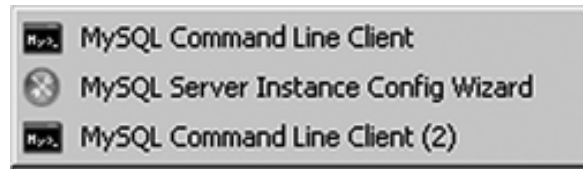


FIGURA 2.27 - TELA DE ATALHO CRIADO

4. Com o mouse, arraste o atalho *MySQL Command Line Client (2)* para área de trabalho:



FIGURA 2.28 - ÍCONE DE ATALHO CRIADO NA ÁREA DE TRABALHO

Muito bem, pelo atalho fica mais prático para você executar o aplicativo *MySQL Command Line Client*, que será usado várias vezes nesta unidade e em outras atividades que serão executadas mais à frente.



Lembre-se que sempre que você executar a chamada do aplicativo acima, será solicitado a senha de acesso, que foi padronizada como **virtual**.

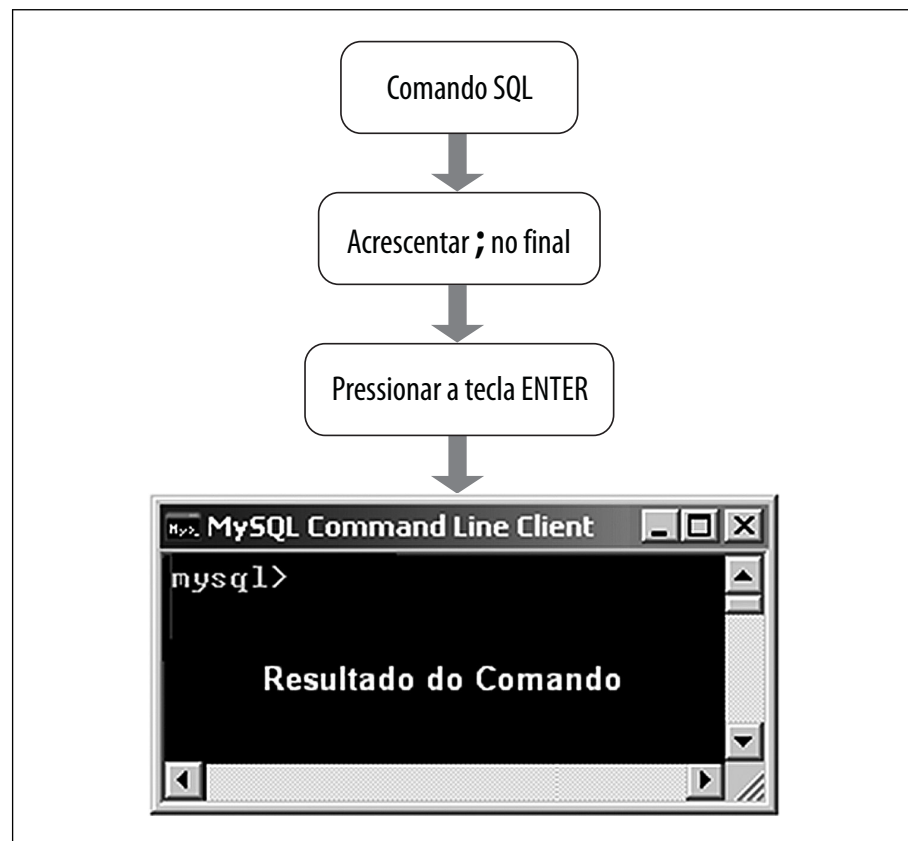
Para começar a interagir com esse aplicativo, clique no atalho criado anteriormente e, na tela que se abre, digite a senha de acesso e pressione a tecla **Enter**.

Não se assuste com esta tela preta, sem graça. Na verdade, essa tela representa o editor de comandos do MySQL e será muito utilizada por você. Esse editor é formado por um cursor que indica a linha atual de edição, representada pelo símbolo *mysql>*.

A partir deste cursor você escreverá o comando que deseja executar, seguido de um ponto e vírgula (;). Para executar o comando, basta pressionar a tecla **Enter**.

Veja o organograma abaixo:

GRÁFICO 2.1 - SEQUÊNCIA DE EXECUÇÃO DOS COMANDOS SQL NO MYSQL COMMAND LINE CLIENT



SEÇÃO 5 – Comandos básicos do MySQL

Agora que você já sabe como executar o editor de comandos do MySQL, está mais do que na hora de começar a praticar, certo? Nesta seção você terá um contato com comandos básicos para a interação entre usuário e MySQL e que são essenciais para que você consiga criar um banco de dados fisicamente.

O ideal é que você pratique os comandos diretamente no editor do MySQL e acompanhe os seus resultados com os que são apresentados aqui na forma de imagens capturadas da mesma ferramenta.

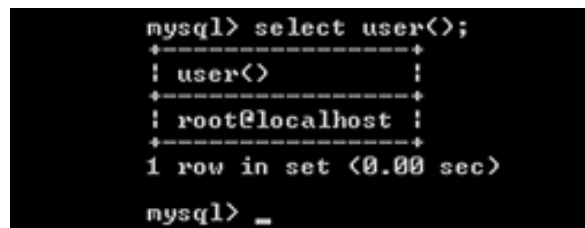
Lembre-se que a tarefa de programar requer um pouco de paciência e muita concentração, principalmente quando o resultado esperado não é alcançado. Mas, o mais importante é que estou nesta caminhada com você.

No primeiro comando, você irá verificar qual é o usuário que está conectado ao banco de dados. Essa é a forma pela qual o sistema de gerenciamento de banco de dados identifica o usuário junto às suas funcionalidades o que ele irá disponibiliza.

Comando Select User

Esse comando apresenta na tela o nome do usuário administrador do banco de dados e o servidor do banco de dados. Como a instalação que foi realizada por você, seguindo o padrão apresentado por mim, foi feita num banco de dados local, a saída do comando será: `root@localhost`.

Para executar o comando, ao lado do prompt **mysql>** edite o comando `select user();` seguido de um ponto e vírgula e pressione a tecla *Enter*. Veja a seguir.



```
mysql> select user();
+-----+
| user() |
+-----+
| root@localhost |
+-----+
1 row in set (0.00 sec)

mysql> _
```

FIGURA 2.29 - RESULTADO DO COMANDO `SELECT USER`

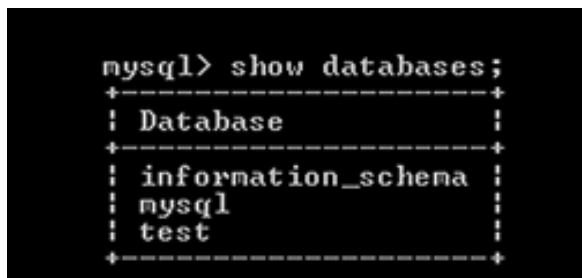
Os comandos podem ser escritos em maiúsculas ou minúsculas, sem distinção pelo aplicativo MySQL.

Comando Show DataBases

Cada conjunto de tabelas do MySQL pode, e deve, ser organizado em banco de dados, chamado de DataBases. Cada DataBase é representado por um nome.

Na instalação do MySQL, alguns DataBases já são instalados. Para verificar quais são esses bancos de dados, digite o comando: *Show Databases; e pressione a tecla Enter.*

O resultado deve ser:



```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schena |
| mysql      |
| test       |
+-----+
```

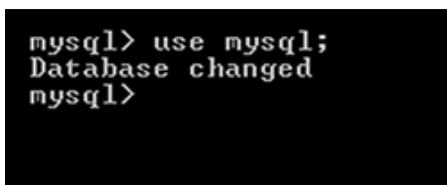
FIGURA 2.30 - RESULTADO DO COMANDO SHOW DATABASES

Comando Use

O acesso ao *database* é realizado por meio do comando **Use** seguido do nome do DataBase, ou seja, a sintaxe do comando é:

Use Nome do DataBase ; <enter>

Veja a seguir o comando para acesso ao DataBase chamado *mysql*: *Use MySQL;*



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

FIGURA 2.31 - RESULTADO DO COMANDO USE MySQL

Comando Select DataBase

Para visualizar o DataBase atual, que está sendo usado, você executou o comando `Select DataBase ()`. O resultado é a apresentação do nome do banco de dados em uso.

Veja a seguir.

```
mysql> select database();
+-----+
| database() |
+-----+
| mysql      |
+-----+
1 row in set (0.00 sec)
```

FIGURA 2.32 - RESULTADO DO COMANDO *SELECT DATABASE*

Comando *Show Tables*

Para visualizar as tabelas que compõem um determinado DataBase, deve-se executar o comando *Show Tables*, da seguinte forma: *Show Tables*; seguido de *Enter*.

Veja abaixo o resultado desse comando, listando as tabelas do DataBase MySQL, uma vez que o comando anteriormente executado foi *use mysql*.

```
mysql> show tables;
+-----+
| Tables_in_mysql |
+-----+
| columns_priv     |
| db               |
| func             |
| help_category    |
| help_keyword     |
| help_relation    |
| help_topic       |
| host             |
| proc             |
| procs_priv       |
| tables_priv      |
| time_zone        |
| time_zone_leap_second |
| time_zone_name   |
| time_zone_transition |
| time_zone_transition_type |
| user             |
+-----+
17 rows in set (0.00 sec)
```

FIGURA 2.33 - RESULTADO DO COMANDO *SHOW TABLES*

Comando Create DataBase

O comando *Create DataBase* cria um banco de dados dentro do MySQL. A criação de um DataBase é fundamental para a organização das tabelas que serão manipuladas por um software.

A conexão do banco de dados com um sistema computacional é realizada por meio do DataBase, que por sua vez, disponibiliza um conjunto de tabelas que podem ser manipuladas. Veja a representação a seguir:

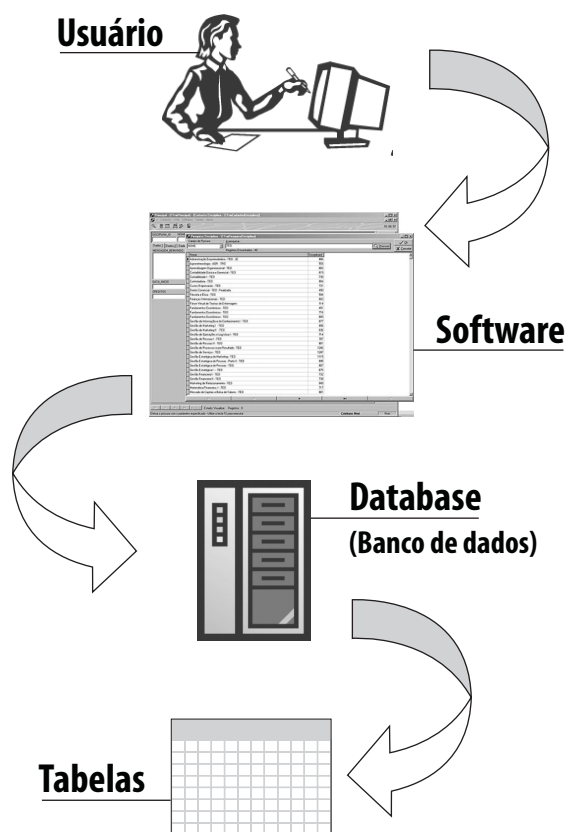


FIGURA 2.34 - REPRESENTAÇÃO DO ACESSO ÀS TABELAS DE UM DATABASE

Para criar um DataBase deve-se executar o comando create DataBase, com a seguinte sintaxe:

```
create DataBase Nome_do_DataBase;
```


Para exemplificar, vou criar o DataBase chamado *SabeTudo*, que deverá representar o banco de dados para o software da escola “Sabe Tudo”, proposto na unidade anterior.

Veja como é simples a sintaxe:

```
mysql> create database SabeTudo;
Query OK, 1 row affected (0.06 sec)

mysql> _
```

FIGURA 2.36 - RESULTADO DO COMANDO CREATE DATABASE SABETUDO;

Para listar os DataBases existentes, você pode executar o comando *show Databases*:

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql      |
| sabetudo   |
| test       |
+-----+
4 rows in set (0.09 sec)

mysql> _
```

FIGURA 2.37 - RESULTADO DO COMANDO SHOW DATABASES;

Você pode notar que foi acrescentado o DataBase chamado *sabetudo* à lista de DataBases do MySQL. A partir deste momento, todas as regras e tabelas serão criadas dentro do DataBase chamado *SabeTudo*, como forma de organização das informações referentes ao software que será implementado.

Vale salientar que o nome do DataBase poderia ser qualquer um, porém a escolha pelo nome *SabeTudo* tem a finalidade de associar o banco de dados ao problema que será solucionado posteriormente.

Como o DataBase criado, o próximo passo é criar as tabelas relacionais que farão parte deste banco de dados. A criação das tabelas se baseia nos comandos de definição de dados do SQL, que são basicamente: *Create Table*, *Drop Table* e *Alter Table*.

Antes de criar este conjunto de tabelas, vou rerepresentar o modelo que foi criado na unidade anterior, que inclusive foi transformado em um *script* de banco de dados.

Veja novamente o modelo proposto para o *software* da escola “SabeTudo”:

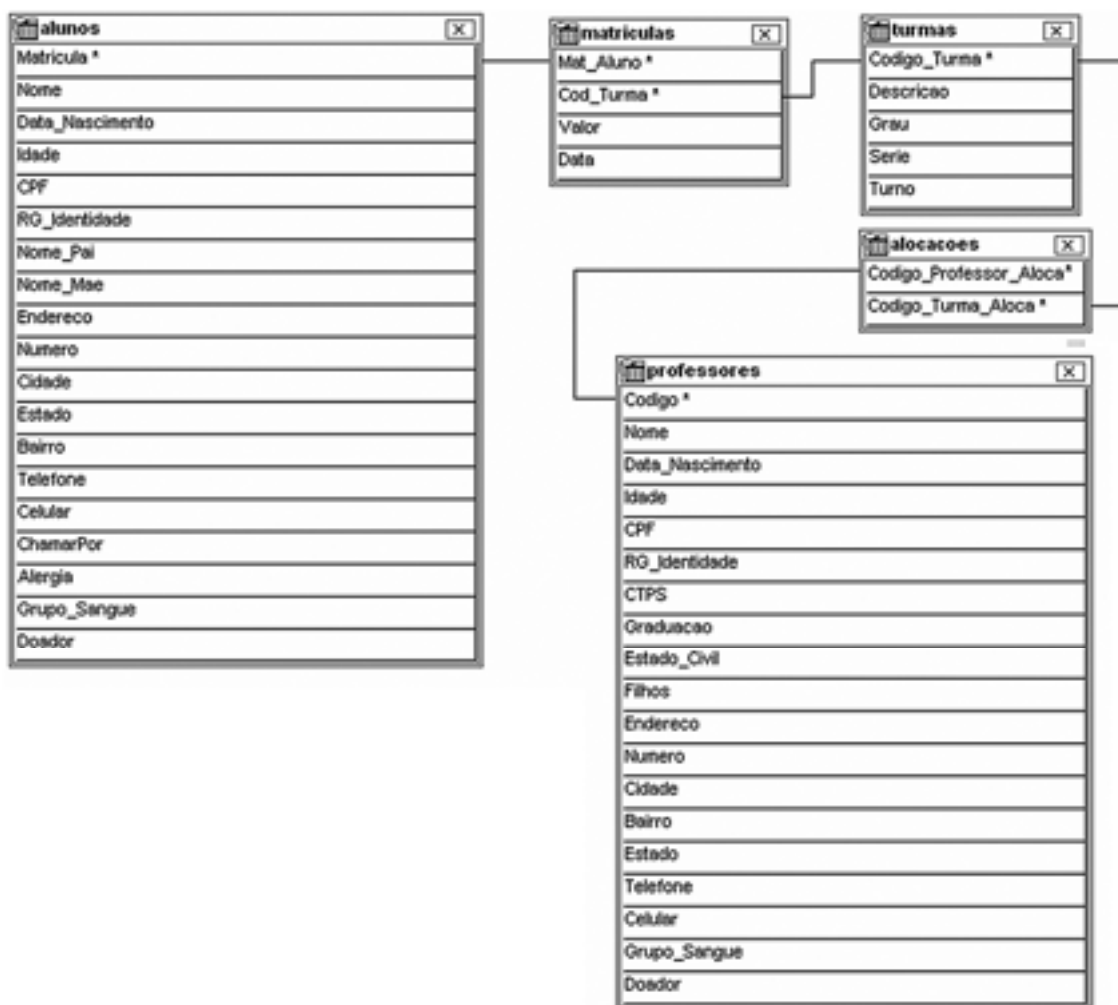


FIGURA 2.37 - MODELO DE DADOS COMPLETO

Assim como o modelo de dados proposto deve ser lembrado, é importante lembrar também o *script* de banco de dados que cria fisicamente esse modelo. Esse *script* possui a sintaxe de criação de cada uma das tabelas, com os nomes e tipos de cada campo, com suas regras de chave primária e chave estrangeira.

A principal finalidade do *script* é ter, em um único arquivo, os comandos que criam as tabelas de um DataBase, de forma organizada e simples. Veja o *script* de criação das tabelas anteriores:

```
-- Script de Criação das Tabelas

-- Remove as Tabelas
DROP TABLE IF EXISTS ALOCACOES;
DROP TABLE IF EXISTS MATRICULAS;
DROP TABLE IF EXISTS ALUNOS;
DROP TABLE IF EXISTS TURMAS;
DROP TABLE IF EXISTS PROFESSORES;

-- Criar a Tabela de Alunos

CREATE TABLE ALUNOS (
  Matricula          Int(11) NOT NULL,
  Nome               Varchar(40) NOT NULL,
  Data_Nascimento    Date NOT NULL,
  Idade              Int(4) NOT NULL,
  CPF                Varchar(14),
  RG_Identidade      Varchar(20),
  Nome_Pai           Varchar(40) NOT NULL,
  Nome_Mae           Varchar(40) NOT NULL,
  Endereco           Varchar(50) NOT NULL,
  Numero             Int(4) NOT NULL,
  Cidade             Varchar(35) NOT NULL,
  Estado             Char(2) NOT NULL,
  Bairro             Varchar(40) NOT NULL,
  Telefone           Varchar(30),
  Celular            Varchar(30),
  Chamarpor          Varchar(30),
  Alergia            Varchar(30),
  Grupo_Sangue       Varchar(10),
  Doador             Int(4),
  PRIMARY KEY        ( Matricula )
);

-- Criar a Tabela de Professores

CREATE TABLE PROFESSORES (
  Codigo             Int(4) NOT NULL,
  Nome               Varchar(40) NOT NULL,
  Data_Nascimento    Date NOT NULL,
```

SEGUE ►

```

Idade                Int(4) NOT NULL,
CPF                  Varchar(14) NOT NULL ,
RG_Identidade        Varchar(20) NOT NULL,
CTPS                 Varchar(20) NOT NULL,
Graduacao             Varchar(30) NOT NULL,
Estado_Civil          Varchar(25) NOT NULL,
Filhos                Int(4)  NOT NULL,
Endereco              Varchar(50) NOT NULL ,
Numero                Int(4) NOT NULL,
Cidade                Varchar(35) NOT NULL,
Bairro                Varchar(30) NOT NULL ,
Estado                Char(2) NOT NULL,
Telefone              Varchar(25) NOT NULL,
Celular               Varchar(25),
Grupo_Sangue          Varchar(10),
Doador                Int(4),
PRIMARY KEY          (Codigo )
);

```

#-- Criar a Tabela de Turmas

```

CREATE TABLE TURMAS (
Codigo_Turma          Int(4) NOT NULL,
Descricao             Varchar(30) NOT NULL,
Grau                  Int(4) NOT NULL,
Serie                 Int(4) NOT NULL,
Turno                 Char(1) NOT NULL,
PRIMARY KEY           (Codigo_Turma )
);

```

#-- Criar a Tabela de Matrículas

```

CREATE TABLE MATRICULAS (
Mat_Aluno             Int(11) NOT NULL,
Cod_Turma              Int(4) NOT NULL,
Valor Float(9,2) NOT NULL ,
Data Date NOT NULL,
PRIMARY KEY (Mat_Aluno,Cod_Turma),
FOREIGN KEY(MAT_ALUNO) REFERENCES ALUNOS(MATRICULA),
FOREIGN KEY(COD_TURMA) REFERENCES TURMAS(CODIGO_TURMA)
);

```

```
-- Criar a Tabela de Alocações

CREATE TABLE ALOCACOES (
Codigo_Professor_Aloca      Int(4) NOT NULL ,
Codigo_Turma_Aloca          Int(4) NOT NULL ,
PRIMARY KEY (Codigo_Professor_Aloca,Codigo_Turma_Aloca ),
FOREIGN KEY (CODIGO_PROFESSOR_ALOCA) REFERENCES
PROFESSORES(CODIGO),
FOREIGN KEY (CODIGO_TURMA_ALOCA) REFERENCES TURMAS(CODIGO_
TURMA)
);
```

SEÇÃO 6 – Convertendo o *script* em um banco de dados

O objetivo da conversão de um *script* em um banco de dados é evitar que o especialista em banco de dados edite um a um o código de criação das tabelas. Isso não estaria errado, funcionaria da mesma forma, porém seria mais trabalhoso.



O ideal, portanto, é executar o *script* na ferramenta de banco de dados.

Para converter esse *script* em um banco de dados, você deve executar os passos seguintes.

1. Em um editor de texto, como bloco de notas do Windows, edite o *script* criado na unidade anterior, sem alteração alguma.

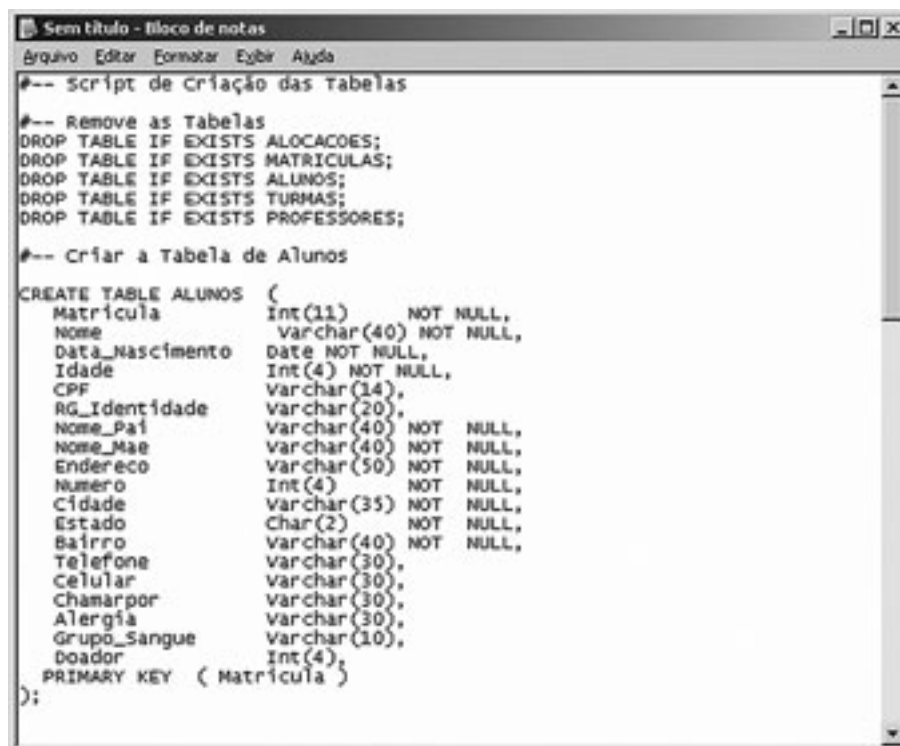


FIGURA 2.38 - SCRIPT EDITADO NO BLOCO DE NOTAS

2. Clique na opção para salvar o arquivo. Redirecione o salvamento para pasta:

C:\Arquivos de programas\MySQL\MySQL Server 5.0\bin.

No formulário **Salvar Como**, na caixa de edição **Nome do arquivo** edite *ScriptSabeTudo.sql* e na opção **Salvar como Tipo** selecione a opção **Todos os Arquivos (*.*)**.

Veja a figura a seguir:

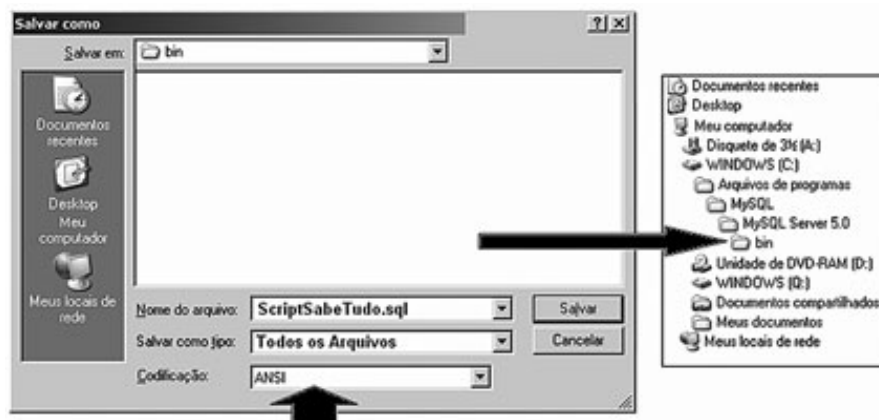


FIGURA 2.39 - SALVANDO O SCRIPT DE CRIAÇÃO DO BANCO DE DADOS

3. Com o *script* salvo, o próximo passo é executá-lo dentro do MySQL. Para isso, carregue o programa *MySQL Command Line Client*. Caso o mesmo não esteja aberto.
4. Pelo comando `Use`, entre no banco de dados *SabeTudo*, executando o comando **`use sabetudo`**.

```
mysql> use sabetudo;
Database changed
mysql> _
```

FIGURA 2.40 - ENTRANDO NO DATABASE SABETUDO;

5. Para executar o *script*, utilize o comando *Source*, da seguinte forma: **`Source ScriptSabetudo.Sql`**;

```
mysql> source scriptsabetudo.sql;
Query OK, 0 rows affected, 1 warning (0.00 sec)
Query OK, 0 rows affected, 1 warning (0.00 sec)
Query OK, 0 rows affected, 1 warning (0.00 sec)
Query OK, 0 rows affected, 1 warning (0.00 sec)
Query OK, 0 rows affected, 1 warning (0.00 sec)
Query OK, 0 rows affected (0.06 sec)
Query OK, 0 rows affected (0.13 sec)
Query OK, 0 rows affected (0.08 sec)
Query OK, 0 rows affected (0.13 sec)
Query OK, 0 rows affected (0.11 sec)
mysql>
```

FIGURA 2.41 - RESULTADO DO COMANDO SOURCE;

6. Caso algum erro aconteça, abra o arquivo *ScriptSabeTudo.sql*, corrija o erro, que normalmente é de escrita, salve a alteração e execute novamente o comando **`source ScriptSabeTudo.Sql`**;
7. Para verificar se as tabelas foram criadas, execute o comando: *show tables*;

```
mysql> show tables;
+-----+
| Tables_in_sabetudo |
+-----+
| alocacoes           |
| alunos              |
| matriculas          |
| professores         |
| turnas              |
+-----+
5 rows in set (0.00 sec)

mysql> _
```

FIGURA 2.42 - TABELAS DO DATABASE SABETUDO

8. Para verificar a estrutura das tabelas criadas, recorra ao comando **Describe**, cuja sintaxe é: desc *nome_da_tabela*. Para ver, por exemplo, a estrutura da tabela de alunos o comando é: *Desc alunos*;

```
mysql> desc alunos;
```

Field	Type	Null	Key	Default	Extra
Matricula	int(11)	NO	PRI		
Nome	varchar(40)	NO			
Data_Nascimento	date	NO			
Idade	int(4)	NO			
CPF	varchar(14)	YES		NULL	
RG_Identidade	varchar(20)	YES		NULL	
Nome_Pai	varchar(40)	NO			
Nome_Mae	varchar(40)	NO			
Endereco	varchar(50)	NO			
Numero	int(4)	NO			
Cidade	varchar(35)	NO			
Estado	char(2)	NO			
Bairro	varchar(40)	NO			
Telefone	varchar(30)	YES		NULL	
Celular	varchar(30)	YES		NULL	
Clamavpar	varchar(20)	YES		NULL	
Alergia	varchar(30)	YES		NULL	
Grupo_Sangue	varchar(10)	YES		NULL	
Doador	int(4)	YES		NULL	

```
19 rows in set (0.03 sec)

mysql> _
```

FIGURA 2.43 - ESTRUTURA DA TABELA DE ALUNOS

Com isso, você já tem um bom caminho percorrido. O banco de dados está modelado e com as tabelas criadas fisicamente. A próxima tarefa é preparar o ambiente de programação que será responsável por implementar o *software* que fará conexão com o banco de dados criado nessa etapa.

Pronto para mais essa tarefa? Então, até a próxima unidade!



Síntese

Você estudou nesta unidade todo o processo de *download* e instalação de uma ferramenta computacional de banco de dados. Por se tratar de uma atividade com várias tarefas envolvidas, pode parecer complicado. Entretanto, à medida que o processo passe a ser executado várias vezes, tende a se tornar simples e objetivo.

A instalação de uma ferramenta de banco de dados é fundamental para a implementação de um sistema com armazenamento de dados, pois representará, fisicamente no

computador, toda a análise do problema transformada em tabelas relacionais.

Cada ferramenta de banco de dados possui um conjunto de comandos únicos, entretanto, a maioria deles deve atender um padrão internacional de comandos, chamados de SQL-ANSI. Esses comandos são independentes do sistema de gerenciamento de banco de dados utilizado.

A criação de um banco de dados se baseia nos comandos de definição de dados da SQL, que são, basicamente, Create Table, Drop Table e Alter Table. Como esses comandos podem sofrer alterações a partir do modelo de dados especificado, ou alterações no próprio andamento do projeto, o ideal é converter as regras de criação do modelo de dados em um *script* SQL.

A forma de carregar esse *script* na ferramenta de banco de dados pode mudar de um modelo para outro, porém a grande maioria deles possui essa funcionalidade.

Uma ferramenta de banco de dados se baseia em dois grupos de comandos:

- comandos proprietários – seguem a sintaxe da ferramenta ; e
- comandos ANSI – SQL – seguem um padrão internacional.

Esses dois grupos abrangem uma gama muito grande de comandos, no entanto, não há necessidade de decorá-los, uma vez que ao serem utilizados, se tornarão de fácil compreensão e utilização.



Atividades de auto-avaliação

1. A utilização de *script* para criação de banco de dados não é obrigatória, no entanto, ele é muito empregado pelos profissionais de banco de dados. Nesse caso, o *script* deve possuir algumas vantagens, você poderia citar algumas?

2. A alteração do modelo de dados deve gerar, conseqüentemente, uma alteração nos *script* de criação do mesmo, pois, caso contrário, o modelo não representará fielmente o banco de dados utilizado. Essa afirmação está correta? Justifique a sua resposta.

3. O MySQL permite a execução de um *script* de criação de tabelas de um banco de dados sem que tenha sido especificado antes qual o DataBase usado? Caso isso não seja possível, como garantir que as tabelas sempre serão criadas no DataBase correto ao se executar o *script*?



Saiba mais

Para aprofundar as questões abordadas nesta unidade, você poderá pesquisar em:

- NIEDERAUER, Juliano. PRATES, Rubens. **MySQL – Guia de referência rápida**. São Paulo: NovaTec, 2005.
- RANGEL, Alexandre. **MySQL - projeto, modelagem e desenvolvimento**. São Paulo: Alta Books, 2005.
- SUHRING, Steve. **MySQL – A Bíblia**. Rio de Janeiro: Campus, 2002.

E consultando os *sites*:

<http://www.mysql.com>

<http://www.mysqlbrasil.com.br>

Ambiente de programação Java



Objetivos de aprendizagem

- Entender o papel da lógica de programação no desenvolvimento de sistemas computacionais.
- Compreender o funcionamento da linguagem de programação Java.
- Aprender a instalar e configurar o ambiente de programação Java.



Seções de estudo

Seção 1 Apresentando o Java.

Seção 2 Instalando o Java.



Para início de conversa

Na unidade anterior você pôde instalar e configurar uma ferramenta de banco de dados responsável pela manipulação e armazenamento dos dados.

Bem, com isso você venceu uma batalha, mas não venceu a guerra.

Foi um grande passo rumo à vitória, mas ainda falta preparar o ambiente para a linguagem de programação que será usada na implementação do sistema com conexão a banco de dados que deverá solucionar o problema da escola “SabeTudo”.

Bem, imagine a tarefa de desenvolvimento de *software* como sendo em três camadas: **interface gráfica, regras de negócio e banco de dados**. A camada de banco de dados, representada pelo banco de dados criado fisicamente, está pronta.

As outras duas camadas, a camada de regras de negócio - que representa as funcionalidades do sistema, ou seja, o que pode e o que não pode ser feito pelo usuário da aplicação - e a camada de interface gráfica - que representa as telas do sistema - ainda carecem de ser implementadas.

Para a camada de banco de dados você utilizou o aplicativo MySQL. Porém, para camada de interface e regras de negócio será preciso utilizar uma linguagem de programação. Sendo assim, nesta unidade você entrará no mundo da programação por meio da linguagem Java, principalmente no que se refere aos comandos de conexão com banco de dados.

Preparado? Então vamos à luta.

SEÇÃO 1 – Apresentando o JAVA

Antes de apresentar a linguagem de programação Java, é preciso que você compreenda o que é uma linguagem de programação.



Uma **linguagem de programação** é uma metodologia de representação das instruções que deverão ser executadas por um computador, obedecendo as suas regras de sintaxe e semântica. Essas instruções são normalmente chamadas de **comandos da linguagem de programação**. A **sintaxe**, de modo sucinto, significa a escrita de uma palavra e a semântica o significado da palavra.

O principal objetivo de uma linguagem de programação é especificar de forma precisa como o computador deverá :

- processar os dados informados por meio das operações de entrada;
- armazenar os dados;
- apresentar os dados processados pelas operações de saída.



A base de qualquer linguagem de programação é a **lógica**.

A linguagem de programação apenas representa a ordem de processamento do computador a partir da lógica especificada pelo programador. Se a lógica estiver errada, o computador a executará da mesma forma, sem avaliar se o resultado esperado está dentro ou não do esperado.

Sendo assim, um bom programador deve ter uma ótima lógica de programação, sem se prender à linguagem de programação utilizada, pois a linguagem é uma formalização da sua forma de solucionar o problema.

Já o inverso não é verdadeiro, um bom programador que domine várias linguagens de programação não terá sucesso nas suas implementações se a sua lógica de programação não estiver bem amadurecida.

O que é o JAVA?

O Java é uma linguagem de programação totalmente orientada a objetos, fabricada e distribuída pela Sun Microsystems.

Na década de 90, a linguagem Java foi projetada para ser usada na área de eletrônica, principalmente no setor de eletrodomésticos, como concorrente da linguagem C. Mas, foi com a popularização da internet que essa linguagem ganhou mercado.

A principal característica da linguagem Java é a sua **portabilidade**. Um programa implementado em Java pode ser executado em qualquer plataforma, seja ela Windows ou Linux, sem alterações. Para que essa portabilidade seja possível, o Java funciona sob uma arquitetura chamada *Java Virtual Machine*, ou simplesmente JVM.

Como isso funciona?

Um programa escrito em linguagem Java, a partir de um editor de texto padrão, é salvo com a extensão **.java**.

O compilador Java, tanto para a plataforma Windows como para a plataforma Linux, verifica se o arquivo atende a todas as regras de sintaxe e semântica da linguagem.

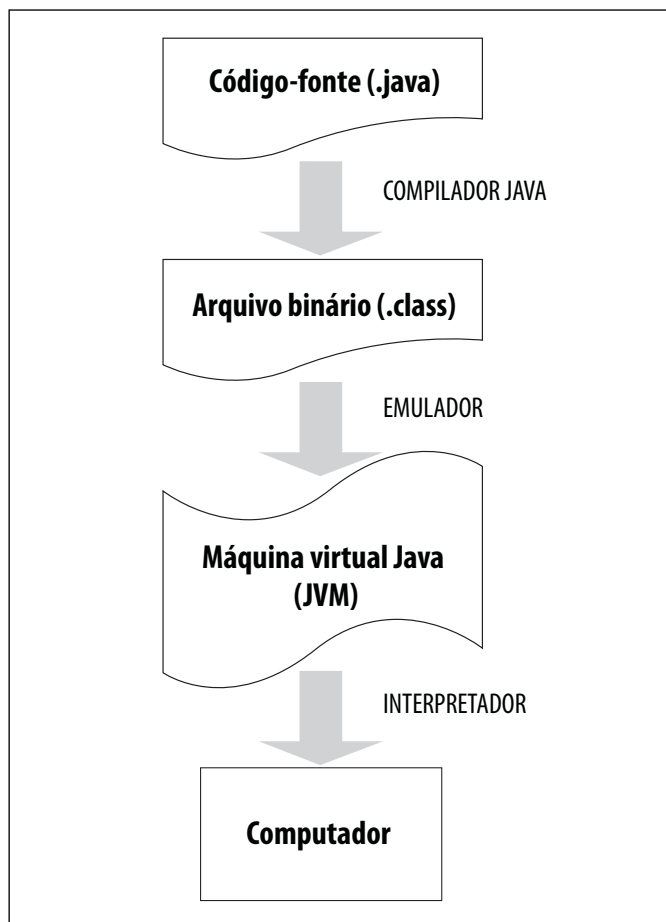
Se não houver nenhum erro, um novo arquivo é gerado, porém com a extensão **.class**.

A partir da geração do arquivo **.class**, o sistema computacional está pronto para ser executado pela máquina virtual do Java, independentemente do sistema operacional do computador. É como se existisse um computador dentro de outro computador.

Dessa forma, o trabalho de executar o aplicativo Java criado é realizado pela máquina virtual usando os recursos do computador no qual a máquina está instalada.

Veja o diagrama a seguir:

GRÁFICO 3.11 - FUNCIONAMENTO DA LINGUAGEM JAVA



O que é preciso para programar em Java

Para programar em Java o usuário precisa instalar na sua máquina um ambiente de desenvolvimento chamado de *Java Development Kit*, ou simplesmente, *JDK*, que pode ser copiado da internet sem custo algum, pois a linguagem de programação Java é gratuita.

A ferramenta Java pode ser encontrada nas plataformas J2E, J2ME e J2EE. Cada uma destas plataformas possui um conjunto de versões e recursos de desenvolvimento.

Para esta disciplina será adotada a plataforma J2SE, na versão 5.0, por atender todas as necessidades do projeto da escola “SabeTudo”.

Nas próximas seções você verá como instalar e configurar o Java no seu computador.

SEÇÃO 2 – Instalando o JAVA

A linguagem Java pode ser obtida no site do seu fabricante, pelo endereço: <www.sun.com>. Por meio desse endereço você deve acessar a área desejada, como Java, downloads e selecionar a plataforma j2e.

— *Um tanto trabalhoso, certo?*

Para facilitar, você pode acessar diretamente a ferramenta que será utilizada nesta disciplina pelo *link*: <<http://java.sun.com/j2se/1.5.0/download.jsp>>. Ao acessar este endereço, clique em **JDK 5.0 Update 6**.

Veja a figura a seguir.

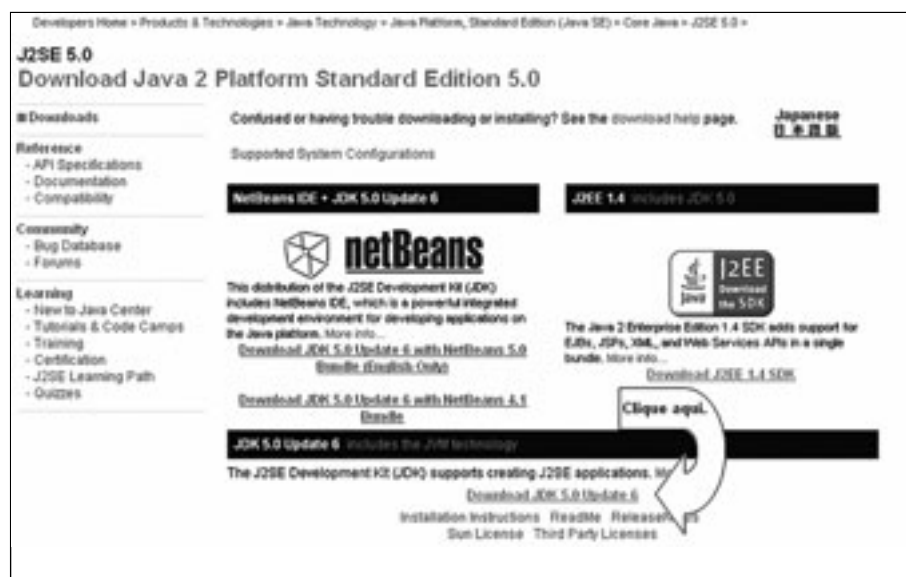


FIGURA 3.1 - LINK PARA DOWNLOAD DA VERSÃO J2SE, 5.0

Na nova página que carregou, marque o item *Required* como *Accept* e depois clique na opção *Windows OffLine Installation Multi Language*.

Veja a imagem a seguir que representa esse processo:

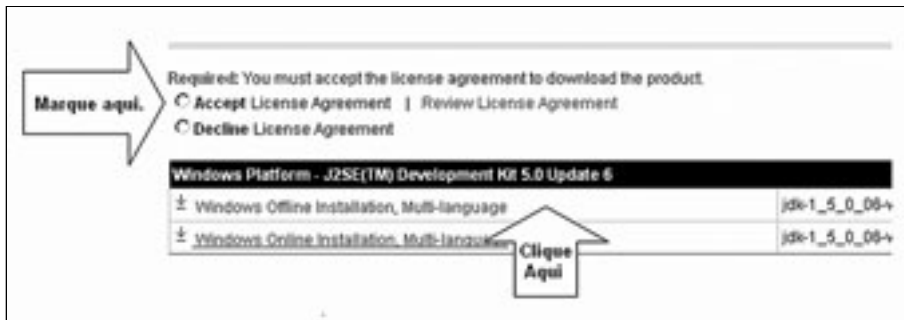


FIGURA 3.2 - TELA DA INSTALAÇÃO DO PROGRAMA

Selecionando a versão para Windows

Na janela que se abriu, clique no botão **Salvar**.

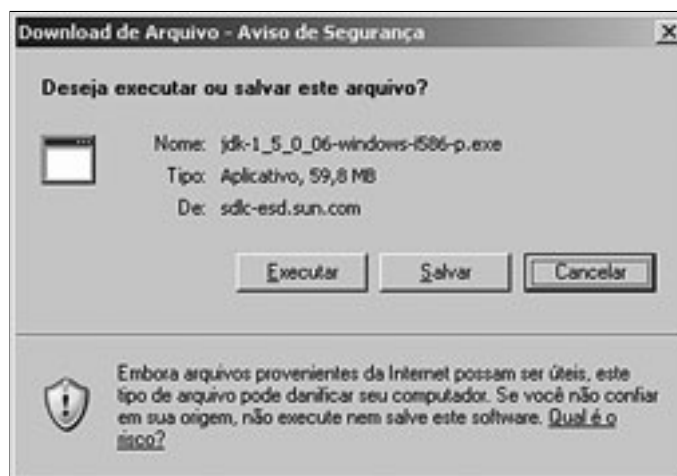


FIGURA 3.3 - TELA DE AVISO DE SEGURANÇA

Carregando o instalador para o computador

Especifique uma pasta para o armazenamento do arquivo e aguarde o *download* ser encerrado.

Após instalar a ferramenta, é necessário realizar algumas configurações no Windows. Essas configurações recebem o nome de **variáveis de ambiente do sistema operacional**, e são fundamentais para o perfeito funcionamento do Java.

Com essas variáveis de ambiente ajustadas, será possível você executar o Java e a sua máquina virtual de qualquer diretório do seu computador.



Como configurar as variáveis de ambiente?

O processo é simples. Primeiro você precisa chegar ao formulário de configuração das variáveis de ambiente para depois adicioná-las ou alterá-las. Para chegar a esse formulário, siga os passos seguintes:

1. Clique com o botão da direita no ícone **Meu Computador** e selecione a opção **Propriedades**:

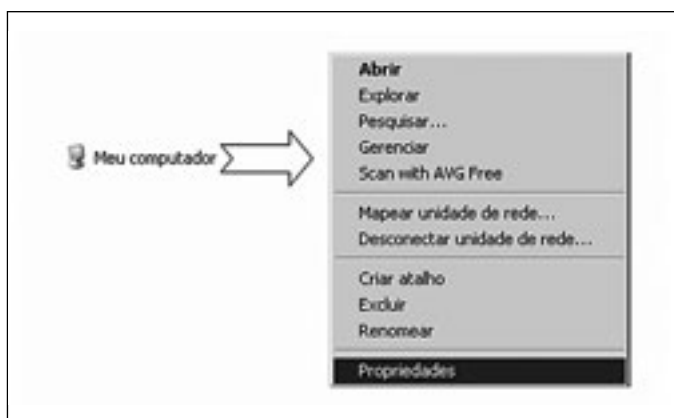


FIGURA 3.4 - PROPRIEDADES DO MEU COMPUTADOR

2. Na janela que se abre, selecione a aba chamada **Avançado** e clique no botão **Variáveis de Ambiente**:

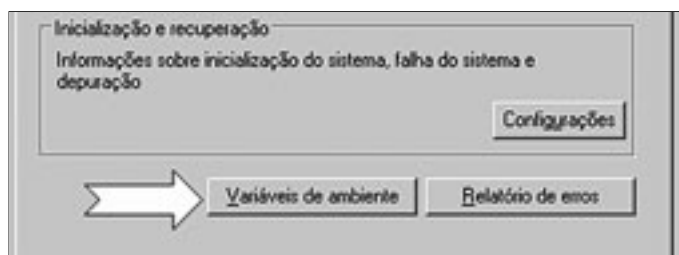


FIGURA 3.5 - ACESSO ÀS VARIÁVEIS DE AMBIENTE

3. Uma nova janela será aberta para a alteração ou inclusão das variáveis de ambiente *CLASSPATH* e *PATH*:



FIGURA 3.6 - VARIÁVEIS DE AMBIENTE



Muito bem, agora você precisa configurar as variáveis de ambiente.

Comece pela variável CLASSPATH, da seguinte forma:

No quadro **Variáveis do Sistema**, na parte inferior do formulário de Variáveis de Ambiente, procure e selecione a palavra *CLASSPATH*. Caso exista esta palavra, e você a tenha selecionado, clique no botão **Editar**:

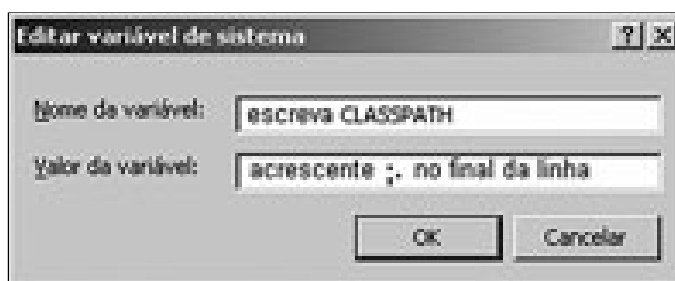


FIGURA 3.7 - ALTERANDO A VARIÁVEL CLASSPATH

No campo **Valor da Variável** apenas inclua no final da linha os símbolos: ; (ponto e vírgula) e . (ponto) e clique no botão **OK**. Dessa forma o campo **Valor da Variável** ficará preenchido da seguinte forma: *Qualquer coisa;*.

Caso a palavra CLASSPATH ainda não exista, clique no botão **Incluir**. A mesma janela acima será apresentada, porém os campos **Nome da Variável** e **Valor da Variável** estarão em branco.

No campo **Nome da Variável** escreva CLASSPATH e no campo **Valor da Variável** insira apenas o símbolo . (ponto) e clique no botão **OK**.

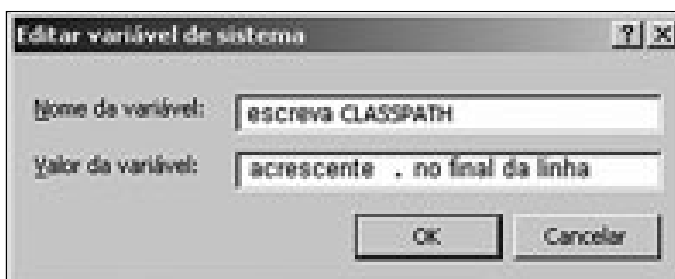


FIGURA 3.8 - INCLUINDO A VARIÁVEL CLASSPATH

Para finalizar, você precisa configurar a variável *PATH*.

Para isso, procure na lista de **Variáveis do Sistema** a palavra *PATH* e caso seja encontrada, selecione-a. Após, clique no botão **Editar**:

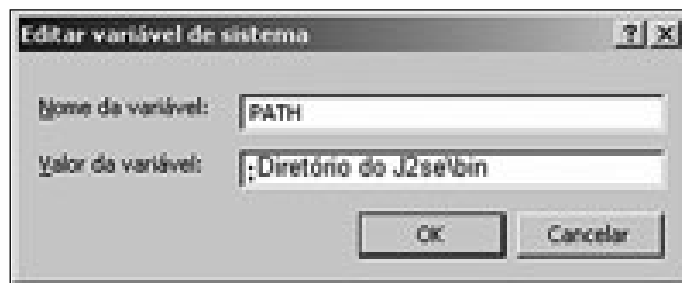


FIGURA 3.9 - CONFIGURAÇÃO DA VARIÁVEL PATH

Antes de editar o valor do campo **Valor da Variável**, verifique no *Windows Explorer* qual o caminho que o Java foi instalado. Por padrão, o Java deve ter sido instalado na pasta: *C:\Arquivos de programas\Java\jdk1.5.0_06*.

Os arquivos executáveis do Java ficam localizados na subpasta chamada **Bin**. São os arquivos **javac.exe** e **java.exe**. Esses arquivos devem ser localizados automaticamente pelo Windows na execução do Java. Para que isso ocorra, é preciso que você insira no campo **Valor da Variável** um ponto e vírgula (;) seguido do caminho: *C:\Arquivos de programas\Java\jdk1.5.0_06\bin*. Depois clique em **OK**.

Caso a variável *PATH* não tenha sido localizada, clique no botão **Nova** e inclua no campo **Nome da Variável** a palavra *PATH* e no campo **Valor da Variável** o caminho: *C:\Arquivos de programas\Java\jdk1.5.0_06\bin*. Depois clique em **OK**.

Para finalizar, clique no botão **OK** da janela de **Variáveis de Ambiente**, voltando à janela de propriedades do sistema. Clique novamente no botão **OK** para fechar esse último formulário.

Para verificar se as alterações das variáveis de ambiente foram feitas de forma correta, você pode executar o Java de qualquer diretório do Windows, da seguinte forma:

- clique no botão **Iniciar** do Windows;
- clique na opção **Executar**;
- no campo abrir escreva **CMD** e clique no botão **OK**.
Será aberta uma janela com o *prompt* do Windows;



FIGURA 3.10 - TELA DE PROMPT DO WINDOWS

Para testar o Java, digite **Javac** e pressione a tecla **Enter**:

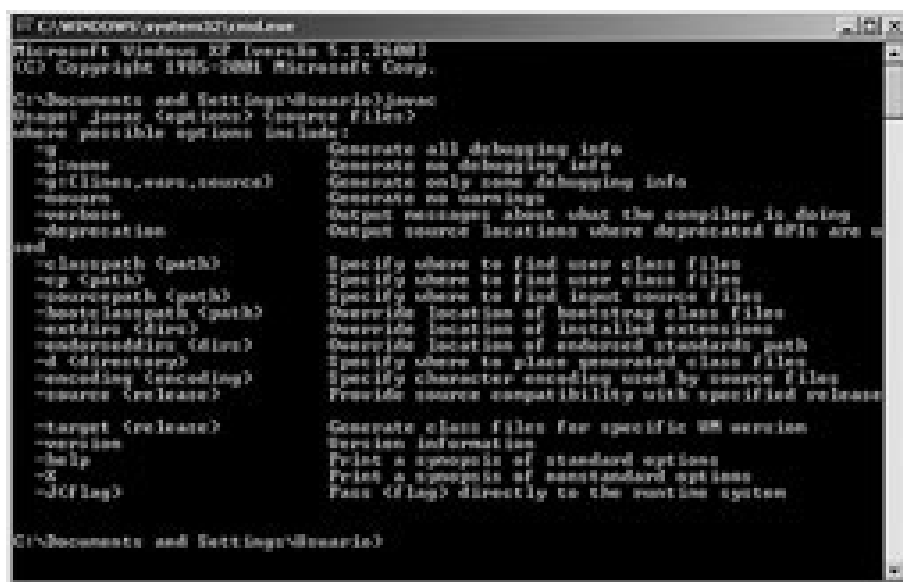
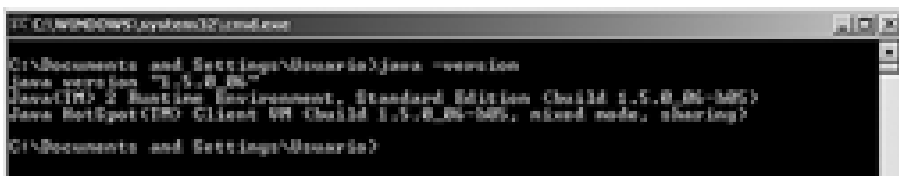


FIGURA 3.11 - RESULTADO DO COMANDO JAVAC



O **comando Javac** faz a compilação de um programa em Java. Quando o comando pode ser executado, isto significa que todas as configurações estão corretas.

Para verificar a versão do Java instalado, digite no *prompt* do Windows a linha de comando: **java -version** e pressione a tecla **Enter**:



```
C:\Documents and Settings\Usuario>java -version
java version "1.5.0_06"
Java(TM) 2 Runtime Environment, Standard Edition Build 1.5.0_06-b05
Java HotSpot(TM) Client VM Build 1.5.0_06-b05; mixed mode, sharing?
C:\Documents and Settings\Usuario>
```

FIGURA 3.12 - VERSÃO DO JAVA INSTALADO



Parabéns! Com isso você obtém a mais uma vitória! Agora você já possui o ambiente de banco de dados instalado e configurado, bem como o ambiente de programação, o Java.

A base para a implementação do projeto para escola “SabeTudo” está quase completa, faltando apenas a conexão entre essas duas ferramentas, mas isso é assunto para a próxima unidade.



Síntese

Nesta unidade você estudou que a lógica de programação é a base de qualquer linguagem de programação.

A linguagem de programação é uma formatação de comandos que serão executados pelo computador, seguindo as regras de sintaxe e semântica. Ela possui duas importantes propriedades: é totalmente orientada a objetos e é gratuita.

Você pôde ainda interagir com todo o processo de “baixar”, instalar e configurar o ambiente de programação Java.



Atividades de auto-avaliação

1. O Java é uma linguagem de programação atual, moderna e muito utilizada. O profissional que é um especialista nessa linguagem de programação possui grandes chances de prosperar nesse mercado de trabalho. Sendo assim, o especialista nesta linguagem não precisa se preocupar, pois estará sempre garantido no mercado, dado que é muito difícil uma linguagem de programação deixar de ser usada, sobretudo a linguagem Java. Essa afirmação está correta? Qual a sua opinião?



Saiba mais

Para aprofundar as questões abordadas nesta unidade, você poderá pesquisar em:

- DEITEL, H.M.;DEITEL,P.J. **Java: como programar**. 6 ed. Ed. Pearson, 2005.
- HORSTMANN, Cay S. **Big Java**. Porto Alegre: Bookman, 2004.

Conectando o Java ao banco de dados



Objetivos de aprendizagem

- Entender o processo de conexão entre uma linguagem de programação e uma ferramenta de banco de dados.
- Compreender o funcionamento do *driver* ODBC e JDBC.
- Instalar e configurar o driver de conexão do Java para MySQL.
- Criar uma fonte de dados de acesso ao banco de dados.



Seções de estudo

Seção 1 *Driver de conexão ODBC.*

Seção 2 *Driver de conexão JDBC.*

Seção 3 *Criando uma Fonte de Dados.*



Para início de conversa

Nas unidades anteriores você pôde preparar o ambiente de desenvolvimento de *softwares*, instalando duas ferramentas fundamentais: o **sistema de gerenciamento de banco de dados** ou simplesmente, banco de dados; e a **linguagem de programação**.

Foram escolhidas duas ferramentas gratuitas, o MySQL e o Java, o que diminuirá em muito o custo de produção do *software* para a escola “SabeTudo”.

A partir de agora é necessário que você compreenda como fazer uma aplicação escrita em Java e se conectar a um banco de dados, de forma segura e confiável.

Sendo assim, seja bem vindo ao mundo dos *drivers* de conexão!

SEÇÃO 1 – *Driver* de conexão ODBC

Uma funcionalidade essencial em qualquer software é a sua **capacidade de se comunicar com um banco de dados**, independentemente da linguagem de programação utilizada para implementação.

A linguagem de programação possui uma sintaxe própria e as ferramentas de banco de dados funcionam baseadas na linguagem SQL – o que significa que essas duas tecnologias não conseguem conversar entre si.

É como se duas pessoas de idiomas diferentes comessem a conversar sem que uma entendesse a outra. Seria um problema, não acha?



Sendo assim, é necessário que um interpretador faça o papel de conversão entre eles, de forma que os dois possam se comunicar.

Veja a imagem a seguir:



Figura 4.1: funcionamento de um *Driver*.

No caso da computação este interpretador recebe o nome de *driver de conexão*. Como cada ferramenta de banco de dados possui particularidades no seu funcionamento, normalmente o *driver de conexão* é distribuído pelo próprio fabricante do banco de dados.



No caso do sistema operacional Windows, há um conjunto de *drivers* para conectividade a banco de dados, chamado de *Open DataBase Connectivity* (Padrão Aberto de Conectividade), ou simplesmente ODBC.

A maioria dos *drivers* de conexão disponíveis no ODBC é para bancos de dados antigos, muitos até deixaram de ser usados, como: Dbase, Clipper. Sendo assim, para a maioria dos bancos de dados atuais é necessário que o usuário instale no seu computador o *driver* específico para a ferramenta que ele está utilizando.

Como o projeto da escola “SabeTudo” se baseia no banco de dados MySQL e na linguagem de programação Java, é preciso que você instale na sua máquina o *driver* de conexão ODBC para o MySQL.



Para isso, acesse o *link* a seguir:

```
<http://dev.mysql.com/get/Downloads/MyODBC3/mysql-connector-odbc-3.51.12-win32.zip/from/http://www.linorg.usp.br/mysql/>
```

Na tela que abrir, clique no botão **Salvar** e direcione o arquivo para uma pasta de sua preferência.

Agora que você “baixou” o arquivo é preciso instalá-lo. Para isso, execute os passos seguintes:

1. Na pasta em que se encontra o arquivo “baixado” anteriormente, dê um duplo clique no arquivo para executá-lo.
2. Na tela do Winzip que se abriu, dê um clique no arquivo chamado **Setup.exe** para iniciar a instalação. Surgirá uma tela, como a imagem a seguir:



Figura 4.2: Tela de Instalação do *Driver* ODBC MySQL.

3. Clique no botão **Next** e repita a mesma tarefa para todas as telas que seguem, até que apareça a seguinte tela:

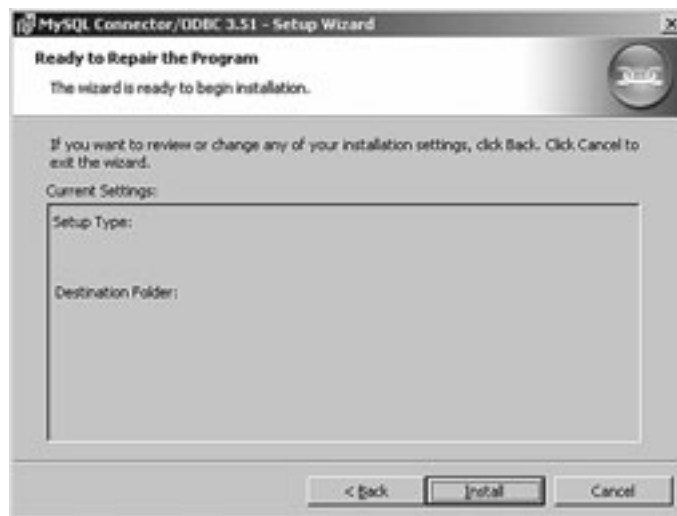


Figura 4.3: Tela de Instalação do *Driver* ODBC MySQL.

4. Clique no botão ***Install*** e aguarde o final da instalação. Assim que a instalação terminar, clique no botão ***Finish*** para encerrar todo o processo.

Pronto, o *driver* ODBC está instalado. A partir desse momento o sistema operacional Windows já reconhece o banco de dados MySQL como uma fonte de dados.

Na próxima seção, você precisará criar uma ponte entre esse *driver* e o *driver* específico para Java, chamado de JDBC. Até lá!

SEÇÃO 2 – Driver de conexão JDBC

No caso do Java, os *drivers* ODBC recebem o nome de JDBC, ou seja, driver ODBC específico para Java. O *driver* JDBC é o *driver* ODBC que permite a conexão entre um banco de dados e o programa escrito em Java. Seria o intérprete entre um sistema desenvolvido em Java e um banco de dados, possibilitando a comunicação entre eles.

Figura 4.4: Papel do *driver* de conexão JDBC.

O uso do JDBC apresenta algumas vantagens para a aplicação Java: **maior portabilidade, facilidade de configuração e padronização.**



Como “baixar” e instalar um *driver* JDBC?

Com base na ferramenta que será utilizada como banco de dados, acesse o *site* do fabricante e procure na área de *download* pelo *driver* de conexão JDBC.

Para facilitar, acesse o *link*: <<http://dev.mysql.com/downloads/connector/j/3.1.html>>, para baixar o JDBC para MySQL. Esse *driver* será usado por você para conexão do *software* para a escola “SabeTudo”. Veja o *site* de *download* desse *driver* de conexão:

Figura 4.5: Site de *download* do *driver* de conexão do MySQL.

Para baixar o arquivo, clique no *link Pick a Mirror*, conforme a imagem a seguir:

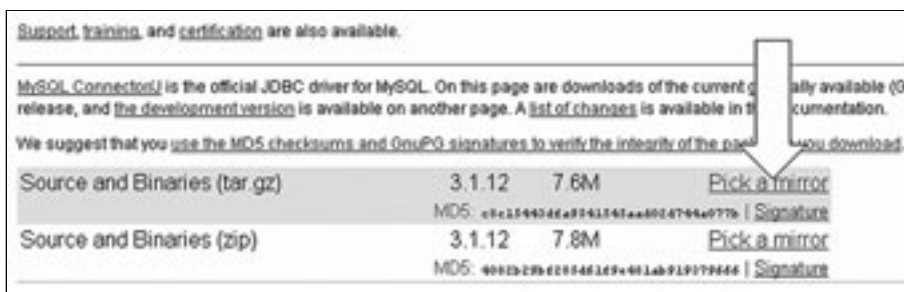


Figura 4.6: Link para o *driver* de conexão do MySQL.

Na página que se abre, procure um Mirror do Brasil e clique na palavra Http, conforme a imagem a seguir:

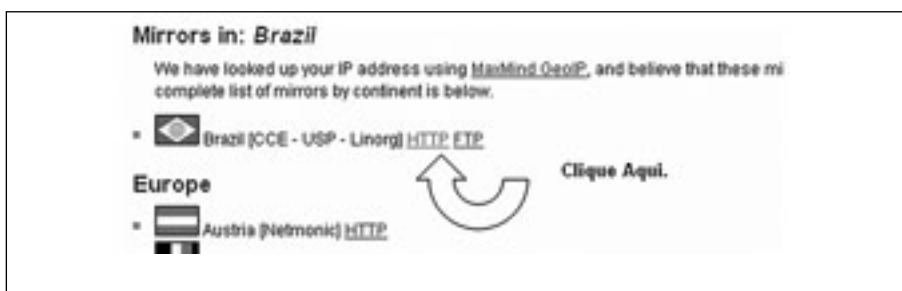


Figura 4.7: Link para iniciar o *download* do arquivo.

Na janela que se abre, clique no botão **Salvar** e direcione o salvamento do arquivo para um diretório de sua escolha.



Figura 4.8: Janela de salvamento do arquivo.

Esse arquivo não precisa ser instalado, basta apenas que ele seja copiado para o computador. Por enquanto é só isso, apenas lembre-se do diretório no qual você salvou esse arquivo, pois na hora da implementação do programa será necessário informar o caminho ao Java.

Como você já realizou várias tarefas, no quadro abaixo são apresentados todas as tecnologias que já foram instaladas por você e a finalidade de cada uma delas, observe:

Quadro 4.1: as tecnologias.

Tecnologia	Situação	Finalidade
MySQL	Instalado e configurado, com a senha de acesso virtual.	Armazenamento dos dados da aplicação para escola "SabeTudo".
Java	Instalado e com as variáveis de ambiente CLASSPATH e PATH configuradas.	Implementar a aplicação para escola "SabeTudo".
JODBC MySQL	Copiado para um diretório do computador.	Conectar o banco de dados MySQL ao aplicativo implementado em JAVA.

Muito bom, com mais essa tarefa você já está com quase todo ambiente preparado para o desenvolvimento da aplicação. Na próxima seção você verá como criar uma fonte de dados ODBC, que, basicamente, significa informar ao sistema operacional algumas características do DataBase que será acessado no MySQL tais como:

- Nome do DataBase acessado;
- Servidor no qual o DataBase está instalado;
- Nome do usuário administrador e a senha de acesso ao DataBase.

Pode parecer complexo, mas você verá que é um processo bem simples.

SEÇÃO 3 – Criando uma fonte de dados

Após a instalação e configuração da ferramenta de banco de dados é necessário que o programador registre no Windows qual o banco de dados será utilizado e onde ele foi criado.

Lembre-se que um banco de dados pode ter sido instalado em uma máquina local, no mesmo computador que estará rodando o programa feito em Java, ou em um servidor, que corresponde a um computador que será acessado por vários usuários.

Esse registro de um banco de dados e seu DataBase no Windows (banco de dados criado fisicamente na ferramenta de banco de dados) recebe o nome de **Fonte de Dados**.

Para criar uma fonte de dados, clique no botão **Iniciar** do Windows e selecione **Configurações | Pannel de Controle**.

Na janela de **Painel de Controle** há um ícone chamado **Ferramentas Administrativas**, clique nesse ícone para que seja aberta a seguinte janela:



Figura 4.9: Tela das ferramentas administrativas.

Para criar uma fonte de dados, clique no ícone chamado **Fontes de Dados (ODBC)**. Será apresentada a seguinte janela na tela:



Figura 4.10: Administrador de fontes de dados ODBC.

Com o formulário aberto, clique na aba chamada **Fonte de dados de sistema**, e depois clique no botão **Adicionar**. Na janela que se abriu, selecione o *driver* de conexão chamado MySQL ODBC 3.51 *Driver*, conforme a próxima figura:

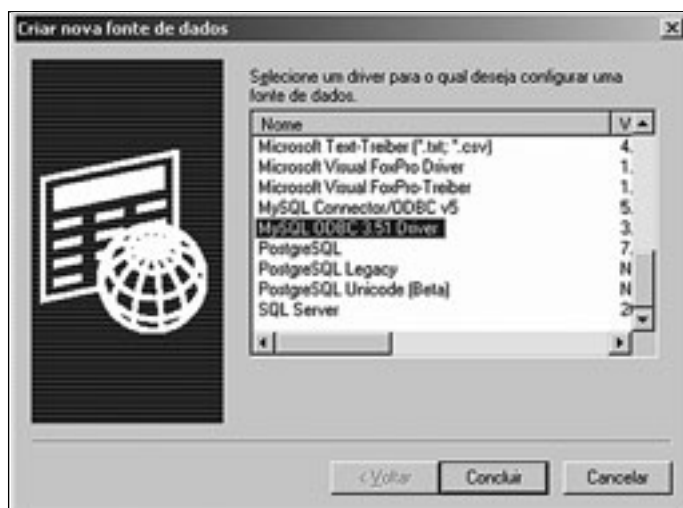


Figura 4.11: *Driver* de conexão do MySQL.

Para finalizar, clique no botão **Concluir**. Será aberta uma janela para criação da fonte de dados que será usada pelo sistema desenvolvido para a escola “SabeTudo”. Nessa janela edite os seguintes valores:

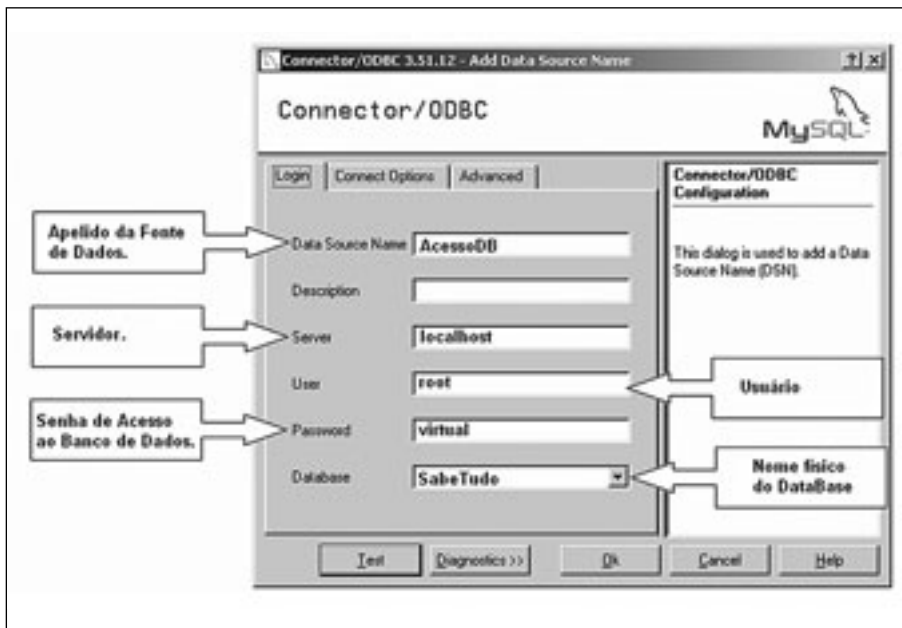


Figura 4.12: Tela do MySQL.

Na caixa de **Apelido da Fonte De Dados**, escreva **AcessoDB**. Este será o nome usado por você toda vez que quiser se referir a essa fonte de dados.

Na caixa **Servidor**, insira a palavra **localhost** para identificar o acesso a um banco de dados local. Caso o banco de dados estivesse em outro computador você informaria o nome ou o IP desse equipamento.

Na caixa de **Usuário**, edite a palavra **root**, que corresponde ao usuário administrador do MySQL.

Na caixa de **Senha**, edite a palavra **virtual** que foi a senha usada por você para a instalação do MySQL.

Para fechar, selecione na caixa de **DataBase** o nome “**SabeTudo**”, que corresponde ao database criado fisicamente no MySQL.

Agora é só clicar no botão **OK**.

O ambiente está pronto para o seu devido uso. Lembre-se que a partir de agora, sempre que você se referir à fonte de dados **AcessoDB**, você está na verdade acessando o DataBase chamado “**SabeTudo**”, através do usuário **root** e senha **virtual**.

Agora é só começar a programar, portanto mãos ao Java. Mas isso é assunto para a próxima unidade!



Síntese

Nesta unidade você pôde estudar o processo de configuração e conexão de um banco de dados com um programa desenvolvido em linguagem Java.

Para que a conexão seja realizada é necessário que o computador possua instalado o *driver* ODBC da ferramenta de banco de dados escolhida. Esse *driver* funciona como um interpretador entre o sistema e a ferramenta de banco de dados, de forma a permitir a comunicação entre ambos.

Um banco de dados possui algumas propriedades, tais como: nome físico, local de instalação, nome do usuário de acesso irrestrito e senha de acesso. Todas essas informações podem ser substituídas por um apelido de conexão, que recebe o nome de fonte de dados.

Sendo assim, a conexão entre o banco de dados MySQL e o Java precisa das seguintes tecnologias: *Driver* ODBC, *Driver* JDBC para o MySQL e uma Fonte de Dados registrada.



Atividades de auto-avaliação

1. Uma característica fundamental num *driver* de conexão com banco de dados é a sua transparência, ou seja, quanto maior for a sua abstração para o programador, melhor a ferramenta. Sendo assim, pode-se afirmar que o programador e o especialista em banco de dados não precisam saber como funciona o *driver*, mas sim, saber como tirar o melhor proveito dele. Esta afirmativa é correta? Justifique sua resposta.

2. O uso das fontes de dados é uma forma de abstração para o programador?



Saiba mais

Para você se aprofundar nesses temas, consulte:

- MECENAS, Ivan. **Fundamentos Swing e JDBC: 2ª edição**. São Pulo: Alta Books, 2005.
- ANSELMO, Fernando. **Tudo que você queria saber sobre JDBC**. São Pulo: Visual Books, 2001.

Ambiente de desenvolvimento integrado



Objetivos de aprendizagem

- Compreender o que é um Ambiente de Desenvolvimento Integrado.
- Instalar e configurar um Ambiente de Desenvolvimento Integrado.
- Compreender o processo de Compilação e Execução de um programa em Java.
- Testar o ambiente Integrado JCreator.



Seções de estudo

- Seção 1** Ambiente de desenvolvimento integrado (IDE).
- Seção 2** Ambiente de desenvolvimento integrado do JCreator.
- Seção 3** Testando o funcionamento do JCreator.



Para início de conversa

Bem, com o Java instalado e configurado, o ambiente de programação está pronto para ser usado. Porém você pode ter a sua vida ainda mais facilitada com a utilização de um ambiente de programação em Java bem amigável, chamado *JCreator*. Esse ambiente, além de fácil de usar, é gratuito.

Sendo assim, a partir de agora você terá acesso a essa ferramenta de programação, como “baixá-la” na internet, instalá-la e configurá-la.

Além disso, você verá alguns comandos em Java específicos para conexão com banco de dados.

Então, até lá.

SEÇÃO 1 – Ambiente de desenvolvimento integrado (IDE)

O Java é a linguagem de programação instalada por você para o desenvolvimento do programa de *software* para escola “SabeTudo”. Isto já seria suficiente para você iniciar a implementação do programa.

O processo seria bem simples, você teria que escrever o seu código em Java em um editor de texto padrão, como o bloco de notas do Windows, salvá-lo com a extensão Java e compilá-lo.

Ufa, parece muita coisa, mas não é.

Veja só, você fará os passos acima de forma prática:

1. Para facilitar, crie uma pasta no seu computador chamada **BD2**. Dentro deste diretório crie uma pasta chamada **Fontes**.
2. Execute o programa bloco de notas do Windows.

- a. Com o editor aberto, edite o seguinte programa em Java:



Figura 5.1 - PRIMEIRO PROGRAMA EM JAVA

3. Salve o arquivo com o nome **ola.java** no diretório **BD2**, na pasta **Fontes**.
4. Agora você vai compilar este programa. Clique no botão **Iniciar**, escolha a opção **Executar**:

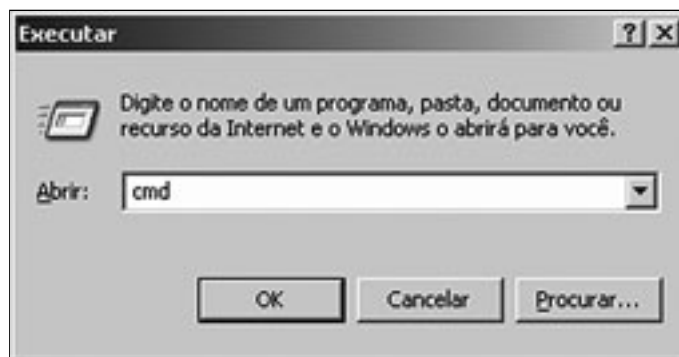


Figura 5.2 - RESULTADO DO COMANDO EXECUTAR

5. Na caixa **Abrir**, edite o comando **CMD** e clique no botão **OK**:

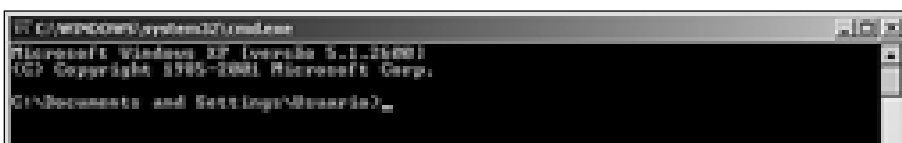


Figura 5.3 - TELA DE *PROMPT* DO WINDOWS

6. Vá para o diretório em que você salvou o arquivo **ola.java**. Algo como:

C:\BD2\Fontes.

7. Execute o comando **dir** para ver se o arquivo **ola.java** está realmente nessa pasta.

```
C:\bd2\fontes>dir
O volume na unidade C é WINDOWS
O número de série do volume é F424-AA48

Pasta de C:\bd2\fontes

14/02/2006 11:03 <DIR>          .
14/02/2006 11:03 <DIR>          ..
14/02/2006 10:51                127 ola.java
                   1 arquivo(s)          127 bytes
                   2 pasta(s) 30.384.238.592 bytes disponíveis

C:\bd2\fontes>
```

Figura 5.4 - RESULTADO DO COMANDO DIR

8. Para compilar o programa, execute o comando **javac** seguido de um espaço em branco em seguida o nome do arquivo que você deseja compilar. Nesse caso, o comando será: **javac ola.java**, não esqueça de pressionar a tecla **Enter** após editar o comando para que o mesmo seja executado.

```
C:\bd2\fontes>dir
O volume na unidade C é WINDOWS
O número de série do volume é F424-AA48

Pasta de C:\bd2\fontes

14/02/2006 11:03 <DIR>          .
14/02/2006 11:03 <DIR>          ..
14/02/2006 10:51                127 ola.java
                   1 arquivo(s)          127 bytes
                   2 pasta(s) 30.384.238.592 bytes disponíveis

C:\bd2\fontes>javac ola.java
ola.java:4: ';' expected
    System.exit(0);
                ^
1 error

C:\bd2\fontes>
```

Figura 5.5 - RESULTADO DA COMPILAÇÃO DO ARQUIVO OLA.JAVA

9. Opa, parece que ao compilar o programa em java, foi encontrado um erro na linha 4, na qual foi esquecido um ponto-e-vírgula. E agora? Simples, você deve esta com o bloco de notas aberto, certo? Caso por algum motivo você tenha fechado, execute-o novamente e abra o arquivo **ola.java** da pasta **c:\BD2\Fontes**.

Na linha 4, desse texto, inclua um `;` no final da linha `System.out.println ("Testando o Java")` e salve novamente o arquivo.



Figura 5.6 - O ARQUIVO OLA.JAVA CORRIGIDO

10. Será preciso, portanto, recompilar o programa, na tela de *prompt* do Windows que ainda deve estar aberta, caso não esteja, repita os processos 5,6 e 7 descritos aqui. Na tela de *prompt* execute o comando `javac ola.java`.



Figura 5.7 - COMPILAÇÃO COM SUCESSO DO ARQUIVO OLA.JAVA

11. Como nenhuma mensagem foi apresentada na tela, a compilação ocorreu com sucesso. Sendo assim, o Java criou um arquivo chamado `ola.class`, que é o resultado da compilação e corresponde ao arquivo que será executado por você. Se você executar o comando `dir`, verá que agora existem dois arquivos chamados `ola`, sendo que um possui a extensão `.java` e outro `.class`.

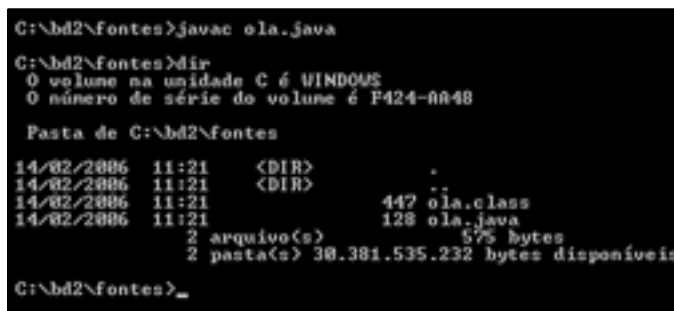
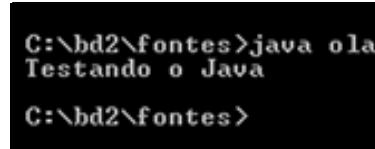


Figura 5.8 - ARQUIVOS DA PASTA C:\BD2\FONTES

12. Para executar o programa (arquivo com extensão .class), na tela de *prompt* do Windows execute o comando **java** seguido de um espaço em branco e o nome do arquivo com a extensão .class, no caso acima o comando ficará: **java ola.**



```
C:\bd2\fontes>java ola
Testando o Java
C:\bd2\fontes>
```

Figura 5.9 - RESULTADO DA EXECUÇÃO DO PROGRAMA OLA

Muito bem, como eu havia escrito anteriormente, pode parecer complicado, mas não é, concorda? É um processo bem trabalhoso, sendo que você precisa trabalhar com a janela do *prompt* e do bloco de notas abertos simultaneamente.

Acredito que seria uma ótima notícia se eu te afirmasse que é possível facilitar em muito todo esse processo. Pois bem, antes de mais nada, feche o bloco de notas e a tela de *prompt* do Windows para que você continue seu estudo.

O Java possui vários programas para edição de seu código-fonte com a finalidade de facilitar o processo de edição e compilação. Poderia citar: NetBeans, Eclipse, JEdit, EditPlus e claro o JCreator.

Todos eles são muito bons, alguns necessitam de um computador mais moderno, principalmente com boa capacidade de memória Ram e são proprietários, ou seja, precisam ser comprados.

No caso do JCreator, além de muito fácil de usar, gratuito, ele é muito leve, não precisa de um computador muito potente para ser executado.

Todos esses facilitadores de implementação para o Java recebem o nome de **IDE**, que vem de ***Integrated Development Environment***, que traduzido fica Ambiente de Desenvolvimento Integrado.

Note que o IDE não é uma linguagem de programação em Java, ele é um complemento de desenvolvimento em Java. Se você não tiver o Java instalado na sua máquina, o IDE de nada servirá.

A principal finalidade do IDE é evitar que o usuário tenha que compilar o programa (por meio do comando `javac`) e executar o programa (por meio do comando `java`) em interfaces diferentes daquela na qual o programa está sendo editado.



Basicamente um IDE deve permitir: compilar o programa, depurar os erros do programa e executar o programa.

Mas como “baixar” e instalar o IDE do JCreator? Isso você verá na próxima seção.

SEÇÃO 2 – Ambiente de desenvolvimento integrado do JCreator

O processo de instalação e configuração do JCreator é bem simples, porém você deve sempre se lembrar que para o devido funcionamento do IDE é necessário que o ambiente Java esteja instalado no computador.

O primeiro passo é “baixar” o instalador da ferramenta para o seu computador. Ao acessar a página de *download* da ferramenta, você deverá cadastrar os seus dados e principalmente o seu *e-mail*, pois você receberá via *e-mail* um *link* para baixar o programa de forma gratuita.



Para baixar o programa, acesse endereço de internet:
<<http://www.jcreator.com/download.htm>>

Para iniciar o processo de *download*, clique no botão de *download* da versão **JCreator LE V3.50 Freeware**.

Na nova página, mude o idioma para português e cadastre os seus dados. Em seguida, clique no botão **enviar**.

Figura 5.10 - TELA DE CADASTRO DO JCREATOR

Será enviado para o seu *e-mail* um *link* para baixar a ferramenta. Não demorará para você receber um *e-mail* com título: *Link para download de JCreator*.



Acesse o endereço enviado para o seu e-mail, algo como: <http://www.jcreator.com/download.php?c=4bf0b74481966caae21ee18d56f628a1>

Salve o arquivo de instalação, chamado **Jcrea350.zip**, em uma pasta de sua preferência.

Para instalar o programa, acesse a pasta na qual você salvou o arquivo e dê um duplo clique no arquivo **Jcrea350.zip**:

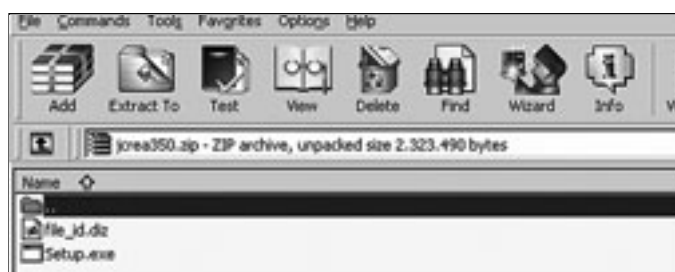


Figura 5.11 - TELA DE INSTALAÇÃO DO JCREATOR

Para iniciar a instalação clique no arquivo **Setup.exe**:



Figura 5.12 - PROGRAMA DE INSTALAÇÃO DO JCREATOR

Para realizar a instalação do JCreator por completo, nas telas que se seguem execute as seguintes tarefas:

1. Na tela 1 - clique no botão *Next*;
2. Na tela 2 - selecione a opção '*I accept de igreement*' e clique no botão *Next*;
3. Na tela 3 - clique no botão *Next*;
4. Na tela 4 - clique no botão *Next*;
5. Na tela 5 - clique no botão *Next*;
6. Na tela 6 - clique no botão *Install*;
7. Para finalizar, clique no botão *Finish*.

Assim que você clicou no botão *Finish*, o JCreator abriu uma janela de configuração, semelhante à figura a seguir. Nessa janela clique no botão *Next* para que o JCreator associe as extensões de arquivos que podem ser editados nele.

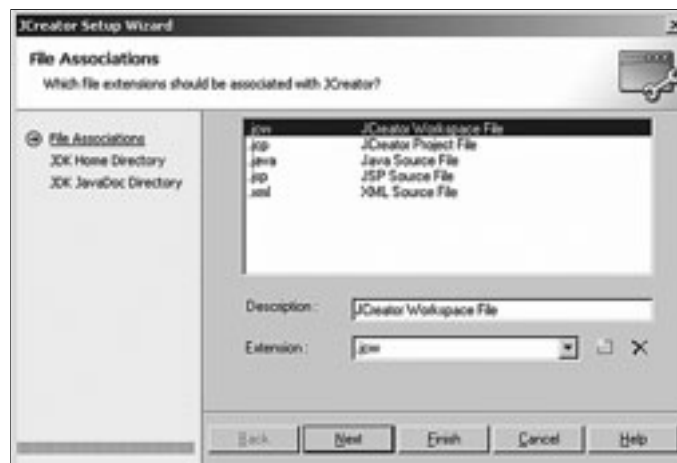


Figura 5.13 - JANELA DE CONFIGURAÇÃO DO JCREATOR

Você deve estar lembrado que aqui mesmo eu escrevi que o IDE é um ambiente de complemento ao Java, sendo que o mesmo só funciona se o computador possuir um ambiente de programação em Java instalado.

Pois bem, na próxima janela o JCreator identifica em qual diretório está instalado o ambiente de programação do Java. Caso o diretório apresentado não seja o mesmo em que você instalou o Java, você pode alterá-lo clicando no botão *Browse*, conforme a imagem que se segue:

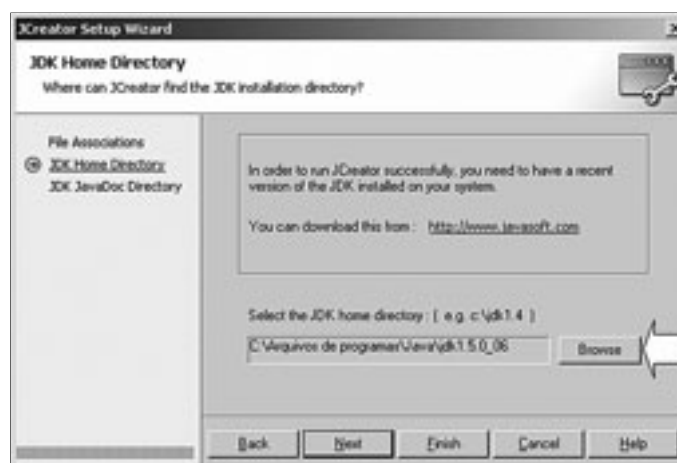


Figura 5.14 - CONFIGURAÇÃO DO DIRETÓRIO PADRÃO DO JAVA

Após confirmar ou alterar o diretório padrão do Java, clique no botão *Finish* para encerrar o processo de configuração. Após isso, o ambiente do JCreator será aberto.

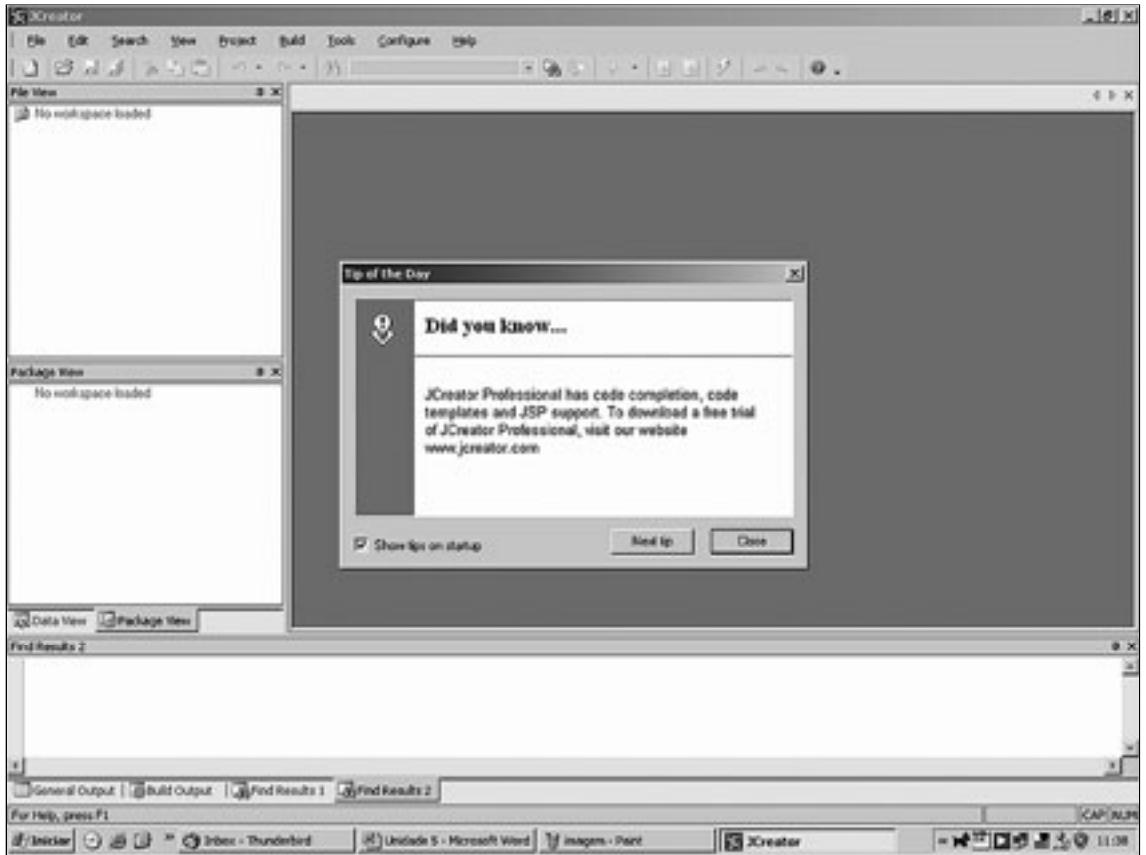


Figura 5.15 - AMBIENTE DE PROGRAMAÇÃO DO JCREATOR

Feche a janela de dicas que aparece no centro da tela. Caso você não queira mais que essa janela seja apresentada a cada vez que você abrir o JCreator, desmarque a opção *'Show tips on startup'* antes de fechá-la.

O uso desse ambiente é muito semelhante à maioria dos editores de texto, sendo assim, acredito que você não terá muitas dificuldades em usá-lo.

Mas isso é assunto para próxima seção.

SEÇÃO 3 – Testando o funcionamento do JCreator

Nesta seção você utilizará o aplicativo JCreator. Para isso é necessário que o ambiente já esteja aberto, caso você precise abri-lo, faça pela opção do grupo de programas do Windows chamado JCreator LE, ou de forma mais rápida, pelo do ícone na área de trabalho.



Figura 5.16 - FORMAS DE ACESSO AO JCREATOR

Para avaliar a ferramenta e testá-la, convido você a editar o mesmo programa em Java feito na primeira seção desta unidade, no qual você usou o bloco de notas e realizou a compilação por meio do *prompt* do Windows.

Primeiro passo, você deverá informar ao JCreator que tipo de projeto deseja implementar. Sendo assim, clique na opção do Menu chamada **File** e escolha a opção **New** e em seguida novamente a opção **File**.

Veja:

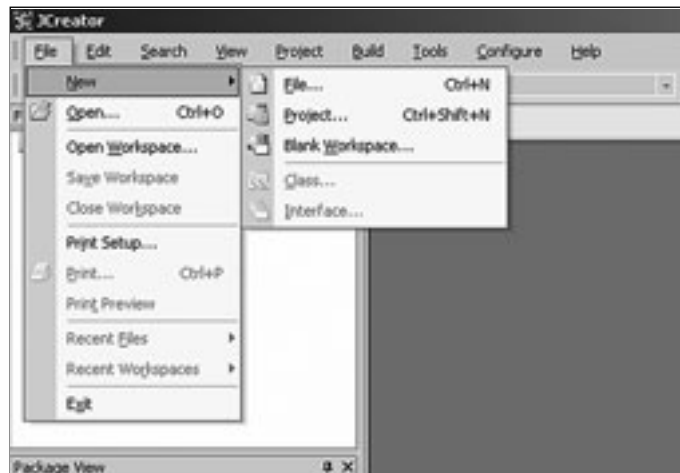


Figura 5.17 - CRIANDO UM NOVO ARQUIVO NO JCREATOR

Serão apresentados a você cinco tipos de arquivos suportados pelo JCreator.

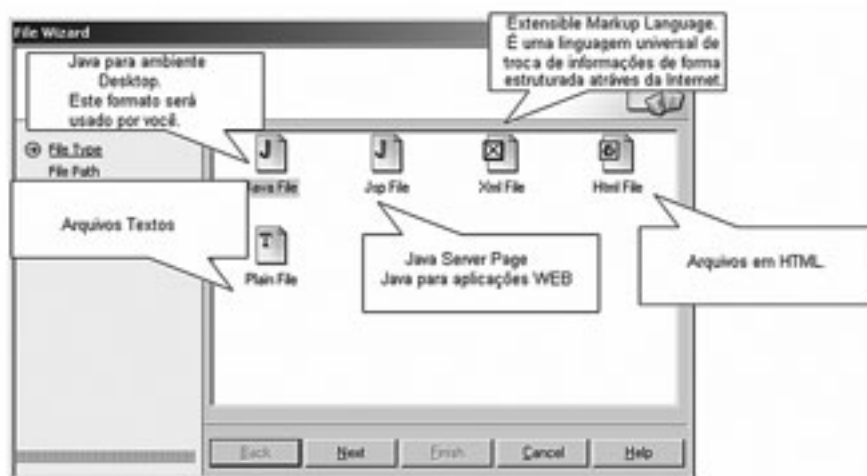


Figura 5.18 - TIPOS DE LINGUAGENS ACEITAS PELO JCREATOR

Selecione a opção *Java File* e clique no botão *Next* para a abertura da janela de configuração do nome do arquivo que será editado.

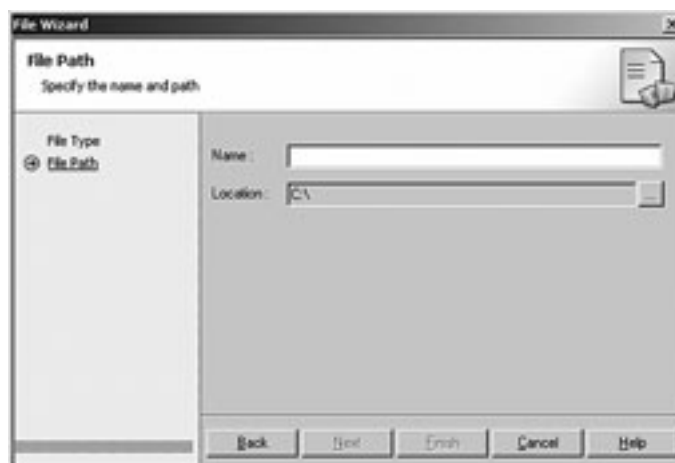


Figura 5.19 - JANELA DE DECLARAÇÃO DO ARQUIVO EM JAVA

Primeira coisa, na caixa *Location*, pelo botão ao lado, direcione a criação do arquivo para o diretório `c:\BD2\Fontes`, criado anteriormente por você.

Na caixa *Name*, edite o nome do arquivo como sendo `ola2`, depois é só clicar no botão *Finish*.

Como resultado, você terá um campo de edição para o arquivo chamado `ola2`.

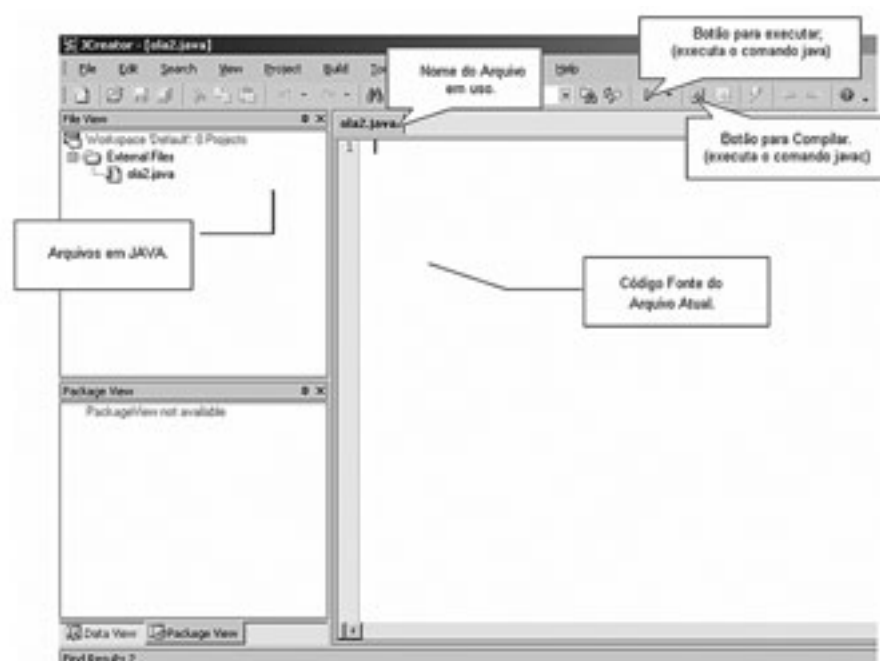


Figura 5.20 - ÁREAS DO AMBIENTE JCREATOR

Para editar o programa de teste, na área de edição do **Código-Fonte do Arquivo Atual**, edite os seguintes comandos:

```
ola2.java *
1 class ola2{
2     public static void main(String args[]){
3         System.out.println("Teste do Java Novamente!");
4         System.exit(0);
5     }
6 }
```

Figura 5.12 - PROGRAMA EM JAVA CHAMADO OLA2.JAVA

Agora que você editou o programa clique no botão de *Compile* para compilá-lo, ou faça por meio da opção de menu chamada *Build*, opção *Compile File*.

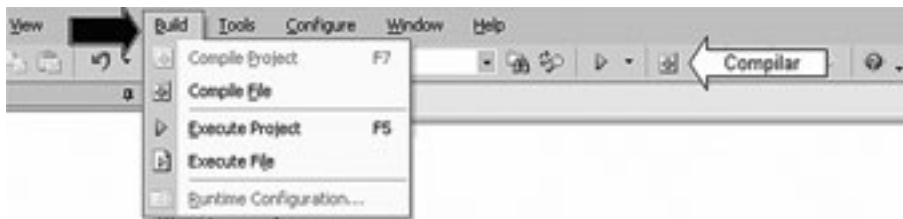


Figura 5.22 - FORMAS DE COMPILAR O PROGRAMA

Opa! O JCreator apresentou um erro de compilação, veja no final da tela que o JCreator abriu uma nova janela, chamada *Task List*.

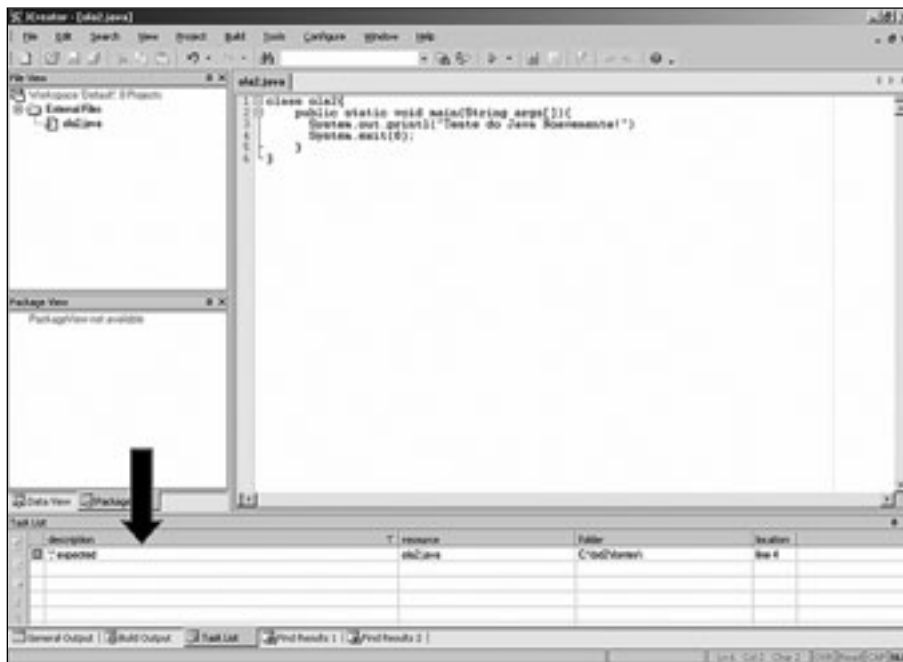


Figura 5.23 - APRESENTAÇÃO DE ERROS DE SINTAXE DO JCREATOR



Mas o que diz esse erro?

A mensagem '**expected**' indica que na linha 4 do código fonte faltou um '**;**'. Para ir direto ao erro, dê um duplo clique na mensagem de erro apresentada na *Task List*.

No código-fonte, inclua no final da linha 3, *System.out.println("Teste do Java Novamente !")* um ponto-e-vírgula (;).

Compile novamente o programa. Se tudo estiver correto a mensagem apresentada pelo Jcreator será:



Figura 5.23 - RESULTADO DA COMPILAÇÃO COM SUCESSO

Para executar o programa, clique no botão de *Execute* ou apenas pressione a tecla **F5**.

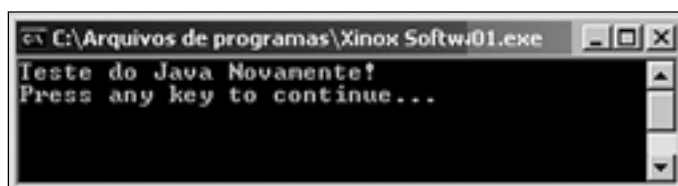


Figura 5.24 - EXECUÇÃO DO PROGRAMA OLA2.JAVA

Com mais esse aplicativo instalado, você está com todo ambiente para o desenvolvimento do programa para escola "SabeTudo" instalado, configurado e principalmente testado.

A partir de agora é hora de você começar a colocar a mão na massa, codificando a solução por meio da linguagem Java.

Preparado para implementação?

Então, até a próxima unidade!



Síntese

Nesta unidade você teve contato com mais um aplicativo essencial para o especialista em banco de dados que deseja implementar um sistema computacional.

Talvez seja o aplicativo mais utilizado por um programador, uma vez que o ambiente de desenvolvimento integrado é responsável por receber toda a codificação do programa em desenvolvimento, o que resumidamente se chama de **código-fonte**.

É importante que você se lembre que o IDE não é uma linguagem de programação e só funciona se houver um ambiente de programação em Java instalado no computador. O aplicativo IDE é um facilitador para o programador, que pode ou não utilizá-lo.

Todo processo de compilação e execução do programa realizado pelo IDE pode ser também executado pelo prompt do Windows por meio de comandos específicos, como *Javac* para compilar o programa e *Java* para executar o programa.

Cabe a você escolher qual a melhor forma de trabalho.



Atividades de auto-avaliação

1. O uso de ferramentas que auxiliam o profissional de banco de dados com interação com a análise de sistemas e programação, as chamadas ferramentas CASE, podem facilitar o processo de implementação a tal ponto que não necessite de um especialista?

2. Seria correto afirmar que o processo de compilação de um arquivo em Java está diretamente relacionada ao código-fonte e a execução diretamente relacionado ao arquivo resultado da compilação chamado de .class?



Saiba mais

<<http://portaljava.com>>

<<http://jcreator.com>>

<<http://www.guj.com.br>>

<<http://www.eclipse.org/>>

<<http://www.netbeans.org/>>

Comandos em Java para conexão com banco de dados



Objetivos de aprendizagem

- Revisar os conceitos básicos da programação em Java.
- Compreender a forma de conexão de uma aplicação ao banco de dados.
- Entender e praticar os comandos em Java que manipulam bancos de dados.
- Implementar regras de banco de dados por meio de classes em Java.



Seções de estudo

- Seção 1** Conceitos fundamentais de programação em Java.
- Seção 2** Conexão ao banco de dados em Java.
- Seção 3** Comandos de manipulação de dados em Java.



Para início de conversa

Amigo de banco de dados,

Agora chegou a hora de você pôr a mão na massa, não há mais como fugir da parte boa da construção de uma aplicação com conexão a banco de dados. Chegou o momento de você começar a implementar algumas regras de negócio da aplicação que solucionará o problema da escola *Sabe Tudo*.

Você já está com o circo armado, então é hora do show.

Acredito que você já tenha tido contato com a linguagem de programação Java, mas mesmo assim, alguns comandos e conceitos serão revisados, o mais importante é que você poderá interagir como novos comandos de conexão ao banco e de manipulação de dados.

Mais do que isso, você irá implementar essas funcionalidades usando a metodologia da programação orientada a objetos e poderá relembrar alguns conceitos importantes dessa metodologia e, principalmente, colocá-los em prática.

Portanto, chega de papo, chegou a hora da codificação.

SEÇÃO 1 – Conceitos fundamentais de programação em Java

Você já teve o contato com a programação em Java por meio da disciplina de Programação Orientada a Objetos, porém é necessária uma revisão de alguns conceitos importantes para o que vem pela frente.

É fundamental que você sempre se lembre que o processo de programar independe da linguagem utilizada, sendo que o mais importante é uma lógica de programação bem estruturada e organizada.

Basicamente, programar abrange as seguintes etapas: **analisar, codificar, compilar, depurar e executar.**



Mas o que é cada uma dessas etapas?

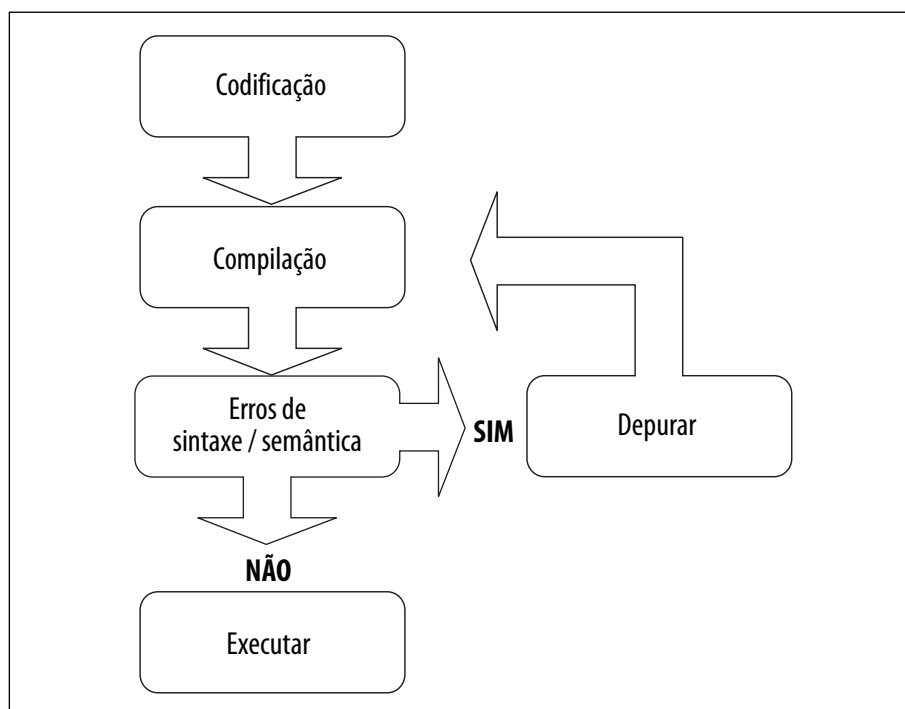
Para responder essa pergunta veja a tabela a seguir.

Quadro 6.1 - ETAPAS DA PRAGRAMAÇÃO

Etapa	Função
Analisar	Converter a solução em um algoritmo.
Codificar	Escrever a solução lógica por meio de uma linguagem de programação.
Compilar	Verificar se o código fonte atende as regras de semântica e sintaxe da linguagem utilizada.
Depurar	Corrigir os erros de sintaxe e semântica apresentados pelo processo de compilação.
Executar	Fazer o programa rodar.

Por meio de um diagrama, o processo pode ser representado da seguinte forma:

Gráfico 6.1 - ETAPAS DO PROCESSO DE PROGRAMAÇÃO



Com base nessas etapas, você estudará o processo de codificação, que está diretamente relacionado com a linguagem de programação Java e à sua estrutura de programação, sintaxe e semântica.

O Java é uma linguagem de programação totalmente orientada a objeto, sendo assim, cada programa implementado em Java é visto como uma **classe**, ou seja, as *regras de manipulação de um objeto*.

O escopo de um programa em Java é:

```
class nome_da_classe {
    Atributos - privados;
    Método Construtor com o mesmo nome da classe e público;
    Métodos Modificadores – públicos;
    Métodos Recuperadores – públicos;
    Métodos das Regras de Negócio da Classe – públicos ou privados;

    Método Main nas classes em que há um método principal.
}
```

Acredito que esse escopo já seja de seu conhecimento.

Veja o exemplo a seguir:

```
import javax.swing.*;
class caneta{
    private String Modelo,Marca,CorTinta;
    private float Preco;

    //-- Construtor
    public caneta(){
        setModelo("");
        setMarca("");
        setPreco(0);
        setCorTinta("");
    }

    //-- Modificadores
    public void setModelo (String _Modelo) {Modelo = _Modelo;}
    public void setMarca (String _Marca) {Marca = _Marca;}
    public void setPreco (float _Preco) {Preco = _Preco;}
    public void setCorTinta(String _Cor) {CorTinta = _Cor;}
```

```

//-- Recuperadores

public String getModelo() { return Modelo;}
public String getMarca () { return Marca;}
public float getPreco() { return Preco;}
public String getCorTinta() { return CorTinta;}

//-- Regras de Negócio

public void LerDados(){
    setModelo(JOptionPane.showInputDialog("Informe o Modelo:"));
    setMarca (JOptionPane.showInputDialog("Informe a Marca:"));
    setPreco(Float.parseFloat(JOptionPane.showInputDialog("Preco:")));
    setCorTinta(JOptionPane.showInputDialog("Cor da Caneta:"));
}

public void ShowDados(){
    JOptionPane.showMessageDialog(null,"Dados da Caneta:\n" +
        "Modelo:" + getModelo() + "\n"+
        "Marca:" + getMarca() + "\n"+
        "Preço:" + getPreco() + "\n"+
        "Cor:" + getCorTinta());
}
}

```

Essa classe implementa as regras de manipulação para um objeto do tipo **caneta**. O usuário do sistema só poderá manipular essa caneta pelos métodos públicos, como o construtor, os modificadores, os recuperadores e os métodos de regras de negócio.



Mas o que se ganha com isso?

Como a classe representa as regras de manipulação do objeto, a partir do momento que você estrutura a sua classe, passa a ter total controle sobre a implementação, pois com certeza conhecerá todas as formas de uso dos objetos resultantes da classe que você criou.

Lembre-se que essa informação é fundamental para você que criou a classe, mas deve ser abstraída pelo usuário do sistema, pois para ele não é importante saber como as classes funcionam ou como foram implementadas.



A classe acima pode ser executada?

A resposta é não. A classe acima implementa as regras da classe **caneta**. Porém, essa classe não possui um método principal, chamado de *main*, e nem é intenção tê-lo.



Mas como executar essa classe?

Para executá-la é preciso que em outra classe em Java seja instanciado um objeto a partir dessa classe.

Ou seja, em outra classe você precisa criar um atributo do tipo **caneta** e a partir disso poderá utilizá-lo por intermédio dos métodos públicos.

Veja:

```
class usacaneta{
    public static void main (String args[]){

        //-- Criando um objeto do tipo caneta
        caneta MinhaCaneta = new caneta();

        //-- Usando o objeto MinhaCaneta
        MinhaCaneta.LerDados();
        MinhaCaneta.ShowDados();

        //-- Sair do Programa
        System.exit(0);

    }
}
```

O atributo **MinhaCaneta** representa a classe caneta. Ao ser instanciado pelo comando *new caneta()*, o usuário poderá utilizar o objeto à vontade, desde que referenciando os métodos públicos. O resultado da execução deste programa será algo como:



Figura 6.1 - ENTRADA DOS DADOS POR MEIO DO MÉTODO LERDADOS()



Figura 6.2 - SAÍDA DOS DADOS POR MEIO DO MÉTODO SHOWDADOS()

Note que para o usuário da classe caneta, ele nem precisa saber como foram implementados os métodos *LerDados* e *ShowDados*, apenas precisa saber usá-los.

Antes de continuar, que tal você editar as duas classes acima no JCreator, gravando-as no diretório **C:\BD2\Fontes**. Depois é só executar a classe **usacaneta** e ver os resultados na tela do seu computador.

Em resumo, podem-se tirar alguns comandos importantes da classe caneta, veja!

Quadro 6.2 - LINHAS DE COMANDO E SUAS FUNÇÕES

Linha de Comando	Função
<code>import javax.swing.*</code>	A importação dessa classe se faz necessária para o uso dos comandos JOptionPane.showInputDialog e JOptionPane.showMessageDialog .
<code>class caneta</code>	A definição do nome da classe deve ser o mesmo nome do arquivo .java .
<code>public caneta</code>	O método construtor deve ter o mesmo nome da classe e deve ser público.
<code>private</code>	Os atributos devem ser privados.
<code>public void</code>	Os métodos modificadores, além de públicos, não retornam valor algum. Além disso, sempre recebe um valor como parâmetro.
<code>public String</code>	Os métodos recuperadores, além de públicos, sempre retornam um valor que deve ser do mesmo tipo do atributo ao qual ele está referenciado. Este método nunca recebe parâmetro.
<code>Float.parseFloat</code>	A entrada de dados via JOptionPane.showMessageDialog sempre retorna uma String . Sendo assim, foi necessário converter um valor do tipo String para o tipo Float .
<code>\n</code>	Caractere especial para pulo de linha.

Os comandos apresentados no quadro não são muito diferentes do que você viu em Programação Orientada a Objetos. O que pode mudar é a forma de representação dos comandos nas classes, mas o processo de codificação é o mesmo, utilizando o que é há de melhor na programação orientada a objetos.

Na próxima seção você incluirá ao seu conhecimento em Java os comandos de conexão e manipulação de banco de dados.

SEÇÃO 2 – Conexão ao banco de dados em Java

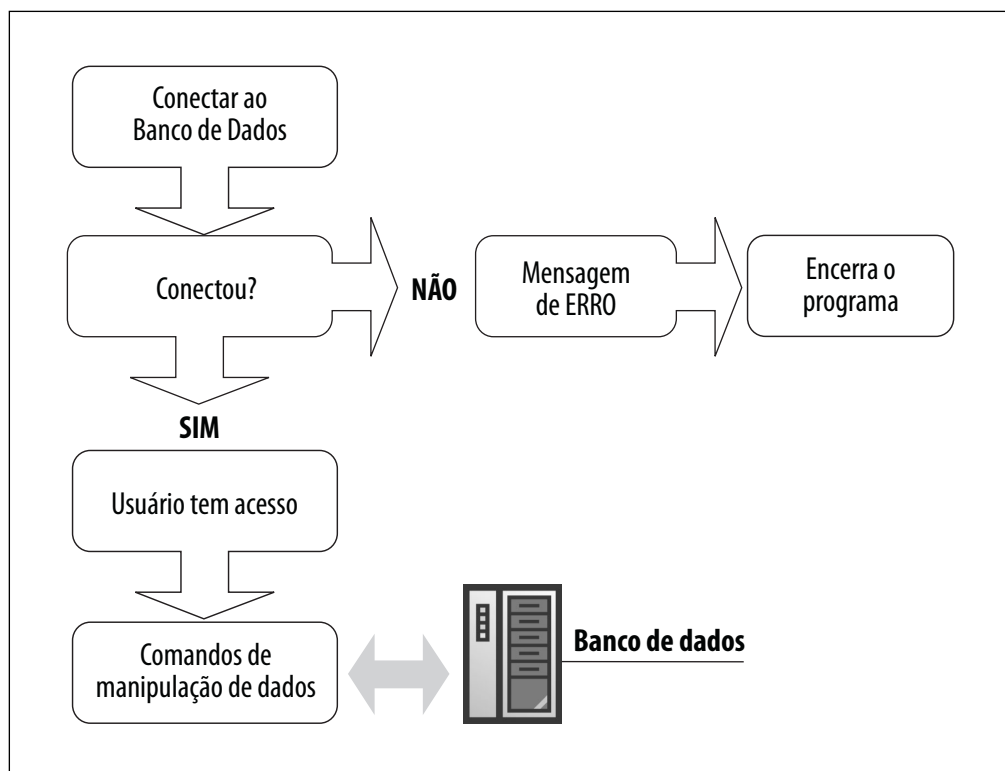
A conexão com um banco de dados, por meio da linguagem Java pode ser dividida em três processos distintos:

- conectar ao banco de dados;
- executar comandos com recuperação de dados – *Select*;
- executar comandos sem recuperação de dados – *Insert*, *Update* e *Delete*.

Sendo que, se o primeiro processo não for realizado com sucesso, os demais não podem ser realizados.

Veja o organograma a seguir:

Gráfico 6.2 - FORMA DE ACESSO AO UM SOFTWARE COM CONEXÃO AO BANCO DE DADOS



A conexão com o banco de dados se dá pela conexão JDBC, utilizando a classe chamada **DriverManager**, que poderia ser traduzido para administrador do *driver*.

Essa classe pertence ao pacote **java.sql** e serve para:

- criar a conexão com o banco de dados;
- administrar o *driver* JDBC usado na conexão;
- controlar o acesso ao banco de dados via login;
- controlar a troca de informação entre o *driver* e a ferramenta de banco de dados.

Como o **DriverManager** pertence ao pacote **java.sql**, sempre que você for utilizá-lo deverá incluir a linha **import java.sql.*** no código-fonte da sua classe, pois esse não é um pacote nativo do Java.



Lembre-se: a linguagem Java é **Case Sensitive**, ou seja, maiúsculo é diferente de minúsculo. Sendo assim, a classe **DriverManager** é reconhecida, já a classe **drivermanager** será um comando não reconhecido pela linguagem.

Como usar a classe DriverManager?

A sintaxe do uso dessa classe é bem simples, como a finalidade dessa classe é especificamente a conexão com o banco de dados, os seus atributos se referem a dados como:

- nome da fonte de dados ODBC;
- nome do usuário;
- senha de acesso ao banco de dados.

Para que a classe **DriverManager** receba esses valores, é necessária a utilização do método **getConnection()**.

Assim a sintaxe de uso do DriverManager é:

```
DriverManager.getConnection(nome da fonte de dados, usuário, senha);
```

Na qual:

- Nome da Fonte de Dados: *String*;
- Nome do Usuário: *String*;
- Senha: *String*;

Os nomes do usuário e da senha de acesso seguem as regras da definição do banco de dados. No caso do banco de dados MySQL instalado por você, seguindo as definições desta disciplina, possui o usuário chamado *root* e a senha *virtual*.

O método `getConnection()` retorna sempre um atributo do tipo *connection*, que pode ser fechado por meio do método `close()`. A partir do momento que o banco de dados foi fechado, ele não pode mais ser manipulado, a não ser que uma nova conexão seja aberta por intermédio do `getConnection()`.

Sempre que você encerrar a aplicação, feche o banco de dados.

Os comandos do `getConnection()` são sempre os mesmos, o que pode mudar são os parâmetros de nome da fonte, usuário e senha.

Sendo assim, a conexão com banco de dados poderia ser representada por uma classe como:

ConexaoDB
- FonteDB: char - Usuario: char - Senha: char
+ Conectar() : boolean + Fechar(): void + Preparar(): void

Figura 6.3 - DIAGRAMA DA CLASSE DE CONEXÃO COM O BANCO DE DADOS EM JAVA

O próximo passo é montar essa classe. Para isso, abra o **JCreator**.

Clique em *File, new File*. Escolha o tipo de arquivo *Java File*.

Nomeie o arquivo como **ConexaoDB**, salve-o na pasta **c:\DB2\Fontes**.

Edite o seguinte código:

```
//--Importe de pacotes
import javax.swing.*;
import java.sql.*;

class ConexaoDB{
    private String FonteDB,Usuario,Senha;
    private Connection Conexao;

    //-- Construtor da Classe com Parâmetros
    public ConexaoDB(String _FonteDB,String _Usuario,String _Senha){
        setFonte (_FonteDB);
        setUsuario(_Usuario);
        setSenha (_Senha);
    }

    //--Modificadores
    public void setFonte (String _FonteDB) {FonteDB = _FonteDB;}
    public void setUsuario(String _Usuario) {Usuario = _Usuario;}
    public void setSenha (String _Senha) {Senha = _Senha;}
    public void setConexao(Connection _Conexao) {Conexao = _Conexao;}

    //-- Recuperadores
    public String getFonte (){return FonteDB;}
    public String getUsuario(){return Usuario;}
    public String getSenha (){return Senha;}
    public Connection getConexao(){return Conexao;}

    //-- Método para conexão com banco de dados
    //-- Retorna verdadeiro ou falso para indicar se a conexão foi feita

    public boolean Conectar(){
        boolean sucesso = true;
        String fonte = "jdbc:odbc:" + getFonte(); //-- Registro da Fonte de Dados
        try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //-- Conexão via ODBC
            setConexao(DriverManager.getConnection(fonte,getUsuario(),getSenha()));
        }
    }
}
```

```

        catch(Exception e){
            sucesso = false;
        }
        if (sucesso)
            JOptionPane.showMessageDialog(null," Conectou com Sucesso.");
        else
            JOptionPane.showMessageDialog(null," Não Conectou.");

        return sucesso;

    }

    //-- Método para fechar a Conexão
    public void Fechar(){
        try{
            getConexao().close();
        }
        catch(Exception e){
            JOptionPane.showMessageDialog(null,"Problema ao Fechar o Banco. ");
        }
    }

    //-- Método para prepara o banco de dados para receber
    // os comandos em SQL.
    public Statement PrepararComando(){
        try{
            return getConexao().createStatement();
        }
        catch (Exception e)
        {
            JOptionPane.showMessageDialog(null," Erro de Comando");
            return null;
        }

    }

}

```

Compile o arquivo, e caso apresente algum erro, compare o que você fez com a listagem apresentada aqui. Vale lembrar que normalmente os problemas podem estar na digitação errada ou em algum símbolo que fora esquecido. Por isso, fique tranquilo!



Mas para que serve essa classe?

Com base nessa classe, é possível se conectar a qualquer tipo de banco de dados, desde que sejam informados a fonte de dados ODBC, a senha e o usuário do banco de dados.

Antes de apresentar mais uma classe, a classe de uso da classe `ConexaoDB`, vou detalhar o código anterior, para que você possa dominar a tecnologia de conexão com banco de dados em Java.

Veja o quadro a seguir.

Quadro 6.3 - LINHAS DE COMANDO E SUAS FUNCIONALIDADES

Linha de comando	Funcionalidade
<code>import javax.swing.*;</code>	Pacote em Java para os comandos derivados de <code>JOptionPane</code> . Como: <code>JOptionPane.showInputDialog</code> e <code>JOptionPane.showMessageDialog</code> .
<code>import java.sql.*;</code>	Pacote Java para manipulação de banco de dados.
<code>class ConexaoDB</code>	Nome da classe criada deve ter o mesmo nome do arquivo em Java.
<code>private String FonteDB,Usuario,Senha; private Connection Conexao;</code>	Atributos da classe são sempre <i>private</i> .
<code>public ConexaoDB(String _FonteDB,String _Usuario,String,_Senha)</code>	Construtor da classe <code>ConexaoDB</code> , recebendo como parâmetro os valores da Fonte de Dados, usuário e senha.
<code>String fonte = "jdbc:odbc:" + getFonte();</code>	O nome da fonte de dados em Java é identificada como "jdbc:odbc:" e em seguida o nome da fonte de dados que se deseja acessar.
<code>Class.forName("sun.jdbc.odbc. JdbcOdbcDriver");</code>	Carrega o driver que será usado para conexão com o Java. Nesse caso, será uma conexão via ODBC.
<code>DriverManager.getConnection(fonte,getUs uario(),getSenha());</code>	Tenta fazer a conexão com o banco de dados a partir da fonte de dados, usuário e senha fornecidos.
<code>setConexao(DriverManager.getConnection (fonte,getUsuario(),getSenha()));</code>	O método <code>setConexao()</code> armazena no atributo <code>Conexão</code> o resultado da execução do comando <code>getConnection</code> .
<code>try{ } catch(Exception e){ }</code>	Toda a operação de banco de dados em Java, obrigatoriamente deve ser protegida com tratamento de exceção try.. catch .
<code>getConexao().close();</code>	Fecha o banco de dados. O método <code>getConexao()</code> retorna a conexão que está aberta e o método <code>close()</code> fecha. Por ser uma operação de banco de dados deve ser protegida por try e catch .
<code>return getConexao().createStatement();</code>	O tipo de dado Statement representa as operações que podem ser executadas pelo banco de dados. Sendo assim, o método PrepararComando mantém o banco de dados em modo de espera, aguardando que algum comando em SQL seja passado para execução.

O mais importante é que essas regras estão prontas, o que pode ser alterado é o nome da fonte de dados, nome do usuário e da senha de acesso. Estes valores são passados para classe `ConexaoDB` no momento em que a mesma é instanciada e a conexão ocorre na chamada do método **Conectar**. Portanto, não há necessidade de reescrever este código de conexão toda vez que for necessário se conectar a um banco de dados.



Lembre-se que esse código permite a conexão a qualquer tipo de banco de dados por meio da conexão via ODBC, basta que sejam informados os dados da conexão.

Mas como utilizar essa classe?

Agora vem o mais importante da orientação a objetos. Para utilizar a classe `ConexaoDB` o programador não precisa saber como que a mesma foi implementada, **basta saber como instanciá-la e usá-la**. Nesse caso, usá-la, significa saber como instanciá-la e como e quando executar os métodos **Conectar**, **Fechar** e **PrepararComando**.

— *Simples não?*

Sendo assim, está na hora de você utilizá-la!

Com o programa JCreator aberto, inclua mais uma classe ao seu projeto, clicando na opção de menu, *File, new File*, e escolha o tipo de arquivo. Nomeie o arquivo para `usarConexao.java` e salve-o na pasta `c:\DB2\Fontes`.

Edite o seguinte código para essa classe:

```
import java.sql.*;

class usarConexao{

    public static void main(String args[]){
        boolean Conectou;
        ConexaoDB C = new ConexaoDB("AcessoDB","root","virtual");
        Conectou = C.Conectar();
        if (Conectou)
            C.Fechar();
        System.exit(0);
    }

}
```

Execute o programa, o resultado deverá ser algo como:

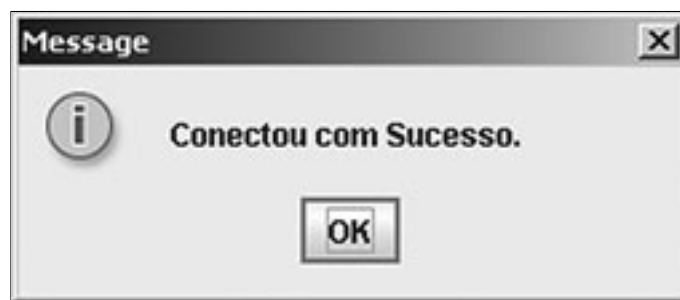


Figura 6.4 - RESULTADO DA EXECUÇÃO DA CLASSE USARCONEXAO

O próximo passo é executar os comandos SQL a partir da conexão com o banco de dados, mas isso é assunto para próxima seção.

SEÇÃO 3 – Comandos de manipulação de dados em Java

Com o banco de dados conectado, o próximo passo é implementar os comandos de manipulação de dados, que são: *Insert*, *Update*, *Delete* e *Select*.

A sintaxe destes comandos você já deve conhecer, portanto, irá apenas revisá-los.

Comando *Insert*

Comando em SQL para o cadastramento dos dados nas tabelas, deve obedecer às regras de chaves primárias, chaves estrangeiras e campos obrigatórios.

Sua sintaxe:

```
Insert into Tabela(Nomes do Campos) values (Valores de cada campo);
```

Comando *Update*

Comando em SQL para atualização dos dados nas tabelas, deve obedecer às regras de chaves primárias, estrangeiras e campo obrigatórios. O comando depende de um filtro para a realização das alterações, caso seja esquecida a condição de filtro, a alteração valerá para todos os registros da tabela.

Sua sintaxe:

```
Update Tabela set Campo = Valor where Condição;
```

Comando *Delete*

Comando em SQL para remoção de dados das tabelas, deve obedecer às regras de chaves estrangeiras. O comando depende de um filtro para a realização das exclusões, caso seja esquecida condição de filtro, a o comando excluía todos os dados da tabela.

Sua sintaxe:

```
Delete from where Condição;
```

Comando *Select*

Comando em SQL para seleção de dados das tabelas, pode referenciar uma ou várias tabelas do banco de dados, tendo a opção de filtrar, agrupar e ordenar os dados.

Sua sintaxe:

```
Select Campos from Tabela  
where Condição  
Group by Campo  
Having Condição  
Order by Campo;
```

Com relação ao Java, o que muda com base nesses comandos é a forma pela qual o comando será repassado para o banco de dados.

Os comandos que não retornam valores, como *Insert*, *Update* e *Delete* são repassados ao banco de dados pelo método *executeUpdate()* da classe *Statement*.



Para remover todos os dados da Tabela de Alunos, os comandos em Java seriam:

```
Connection C = DriverManager.getConnection("fonte","usuario","senha");  
Statement s = C.createStatement();  
s.executeUpdate("Delete from Alunos");
```

Já o comando *Select* é um comando de retorno de dados, sendo assim ele é repassado ao banco de dados pelo método *executeQuery()* da mesma classe *Statement*.

Por exemplo, para listar todos os dados da Tabela de Alunos, os comandos em Java seriam:

```
Connection C = DriverManager.getConnection("fonte","usuario","senha");  
Statement s = C.createStatement();  
s.executeQuery("Select * from Alunos");
```


Como a diferença entre as duas formas de execução dos comandos em SQL é mínima, o ideal é que se crie uma classe para conexão e envio de comandos para o banco dados.

Assim como há uma classe que atende as necessidades de conexão com banco de dados, a classe `ConexaoDB`, é necessária também uma classe para padronização do envio de comandos ao banco de dados.

Portanto, a partir desse ponto, você vai começar a montar a classe de comandos em SQL.

A estrutura da classe será a seguinte:

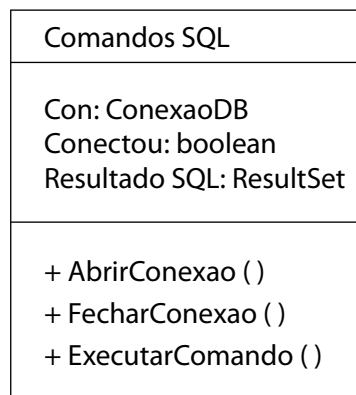
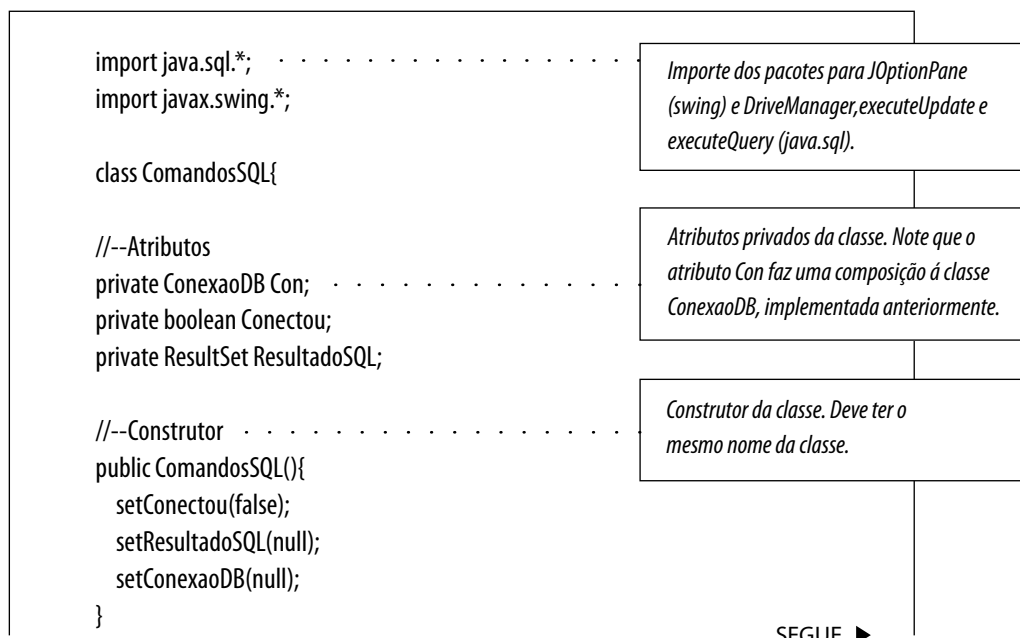


Figura 6.5 - DIAGRAMA DA CLASSE COMANDOSSQL

A seguir é apresentado o código que implementa essa classe:



SEGUIE ►

Este método é fundamental para a execução dos comandos em SQL. Ele abre o banco de dados a partir dos dados de fonte, login e senha.

Fecha a conexão com banco de dados, desde que a abertura do banco tenha ocorrido com sucesso. O método `getConectou()` informa se o banco está aberto.

*Este método é o ponto-chave desta classe. Ao executar um comando em SQL passado como parâmetro, a classe verifica primeiro se o banco de dados está aberto. Se estiver aberto, o próximo passo é verificar se o comando contém a palavra 'select', se contiver, é executado o método `executeQuery()`, que retorna um **ResultSet**, senão executa um `executeUpdate()` que não retorna dados.*

```
//--Modificadores
public void setConectou (boolean _Conectou){Conectou = _Conectou;}
public void setResultadoSQL(ResultSet _Resultado){ResultadoSQL = _Resultado;}
public void setConexaoDB(ConexaoDB _ConexaoDB) {Con = _ConexaoDB;}

//-- Recuperadores
public boolean getConectou(){return Conectou;}
public ResultSet getResultadoSQL(){return ResultadoSQL;}
public ConexaoDB getConexaoDB() {return Con;}

//-- Usa a classe ConexaoDB para abrir o banco de dados
public void AbrirConexao(String _Fonte,String _Usuario,String _Senha){
    setConexaoDB(new ConexaoDB(_Fonte,_Usuario,_Senha));
    setConectou(getConexaoDB().Conectar());
}

//-- Fecha o banco de dados a partir da classe ConexaoDB
public void FecharConexao(){
    if (getConectou()){
        getConexaoDB().Fechar();
    }
    else{
        JOptionPane.showMessageDialog(null,"Banco de dados não estava aberto");
    }
}

//-- Executa o comando em SQL e armazena o resultado do comando quando
//for um select.
public void ExecutarComando(String ComandoSQL) throws SQLException{
    if (getConectou()){
        Statement s= getConexaoDB().PrepararComando();
        if (ComandoSQL.indexOf("select") != -1 ){
            setResultadoSQL(s.executeQuery(ComandoSQL));
        }
        else{
            setResultadoSQL(null);
            s.executeUpdate(ComandoSQL);
        }
    }
    else{
        JOptionPane.showMessageDialog(null,"Banco de Dados Fechado.");
    }
}
}
```

Compile a classe mostrada anteriormente e caso seja encontrado algum erro de sintaxe, compare o seu código com o código-fonte acima. Com essa classe implementada, você já pode passar qualquer comando em SQL para o banco de dados.

No caso dos comandos que não resultam dados, como *Insert*, *Update* e *Delete*, essa classe atende todas as suas necessidades. Porém para o comando de *Select*, que retorna os dados da pesquisa é preciso formatar a apresentação dos dados.

Mas isso é assunto para a próxima classe que você implementará. Antes é preciso que você implemente um programa de teste para classe *ComandosSQL* para testá-la e, mais importante, ver como a mesma funciona.

Para isso, abra o JCreator, crie um novo arquivo do tipo *Java File*, chamado *usarComandos.java*.

Lembre-se de salvá-lo na pasta **c:\BD2\Fontes**.

Edite o seguinte código-fonte:

```
import javax.swing.*;

class usarComandos{
public static void main(String args[]){
    String Titulos[] = {"Código","Descrição","Grau","Série","Turno"};
    String Valores[] = new String[5];
    String LinhaSQL = "insert into turmas values(";
    ComandosSQL com = new ComandosSQL();
    try{
        com.AbrirConexao("AcessoDB","root","virtual");
        if (com.getConectou()){
            for (int i=0; i< 5;i++){
                Valores[i]=JOptionPane.showInputDialog("Informe o " + Titulos[i]+" :");
                Valores[i] = ""+Valores[i]+"";
                if ( i < 4) Valores[i] =Valores[i]+" ";
                LinhaSQL = LinhaSQL + Valores[i];
            }
            LinhaSQL = LinhaSQL + ')';
            JOptionPane.showMessageDialog(null,LinhaSQL);
            com.ExecutarComando(LinhaSQL);
        }
        }catch(Exception e){
            JOptionPane.showMessageDialog(null,"Problema:" + e.getMessage());
        }
        System.exit(0);
    }
}
```

O Vetor *Titulos* representa cada campo da tabela de Turmas do banco de dados MySQL, database chamado *sabetudo*.

O Vetor *Valores* armazena o valor digitado pelo usuário, na ordem dos campos da tabela *Turmas*.

A variável *LinhaSQL* representa o comando SQL que será executado pelo banco de dados.

Comentando o código anteriormente citado, o objetivo dessa classe é cadastrar uma turma no banco de dados. O atributo **Titulos** é um vetor de constantes que representa cada um dos campos da Tabela de Turmas. Cada valor do vetor é apresentado ao usuário na entrada de dados por meio do comando:

```
Valores[i]=JOptionPane.showInputDialog("Informe o " + Titulos[i]+" :");
```

Dessa forma, o vetor chamado Valores recebe o valor de cada campo da inserção, na ordem de cadastramento do usuário.

Tem-se, então, duas estruturas:

	0	1	2	3	4
Titulos	Matrícula	Descrição	Grau	Série	Turno
Valores					

Figura 6.6: ESTRUTURAS USADAS NA CLASSE USARCOMANDOS

Na execução da classe, o valor de cada título informado é cadastrado na sua posição respectiva no vetor de valores. O comando que garante esse funcionamento é:

```
for (int i=0; i< 5;i++){
    Valores[i]=JOptionPane.showInputDialog("Informe o " + Titulos[i]+" :");
```

O resultado da execução desse comando será a solicitação de cadastro de cada campo da Tabela de Turmas, da seguinte forma:

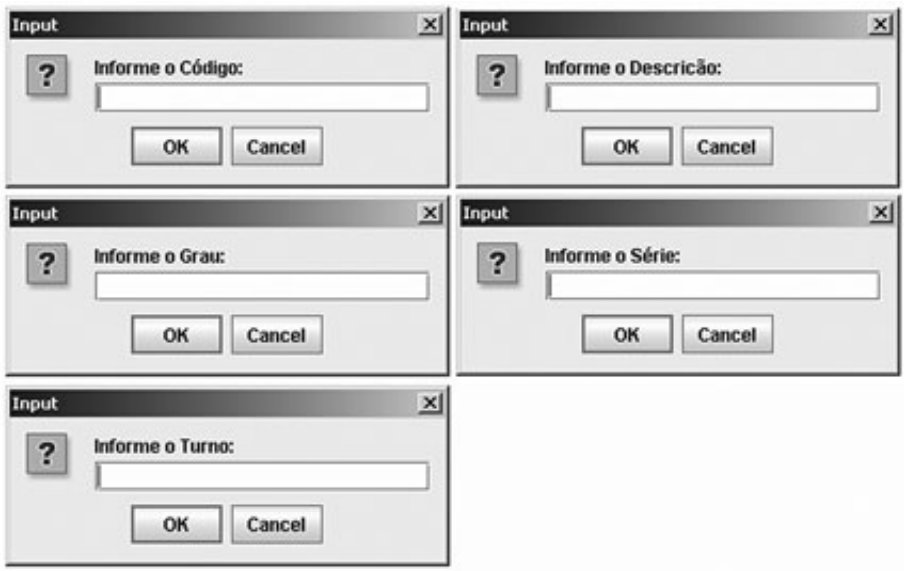


Figura 6.7 - PROCESSO DE ENTRADA DOS DADOS DA CLASSE USARCOMANDOS

Imagine que durante o cadastro, o usuário informou os valores:

Código	Descrição	Grau	Série	Turno
1	Sexta Serie B	1	6	M

Dessa forma, a estrutura usada pela classe ficará assim:

	0	1	2	3	4
Títulos	Matricula	Descrição	Grau	Série	Turno
Valores	1	Sexta série B	1	6	M

Figura 6.8- ESTRUTURA DA CLASSE USARCOMANDOS PREENCHIDAS

O próximo passo é converter essas estruturas para um comando SQL, por isso o uso do atributo LinhaSQL, que foi inicializado com o valor “Insert into Turmas values (“.

Para montar o valor do atributo `LinhaSQL`, as seguintes regras foram implementadas:

- inclusão do símbolo de aspas simples (') antes e depois de cada valor de campo pelo comando:

```
Valores[i] = "'" + Valores[i] + "';
```

- inclusão de uma vírgula (,) após cada valor do vetor de `Valores`, exceto o último campo, de índice 4, que não recebe vírgula, pelo comando:

```
if ( i < 4) Valores[i] =Valores[i]+",";
```

- concatenação dos valores do vetor de `Valores` na *String* `LinhaSQL`.

Imagine essas regras na estrutura de valores da figura anterior, o vetor de valores ficaria assim:

Valores	"1",	"Sexta série B",	"1"	"6"	M
----------------	------	---------------------	-----	-----	---

Figura 6.9 - VETOR DE VALORES COM OS VALORES COM ASPAS E VÍRGULA

Para visualizar a estrutura do comando em SQL criado, foi incluída a instrução:

```
JOptionPane.showMessageDialog(null,LinhaSQL);
```

Cujo resultado é algo como:

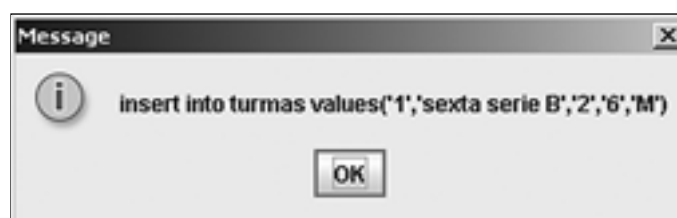


Figura 6.10 - COMANDO EM SQL MONTADO PELA CLASSE USARCOMANDOS

O último passo é enviar o comando para o banco de dados e isto foi feito através da instrução: **com.ExecutarComando(LinhaSQL);**

Com isso o registro informado estará cadastrado no banco de dados, na Tabela de Turmas.

Como você pode ver, são vários detalhes da classe **usarComandos**, por isso é muito importante que você a codifique, compile e execute.

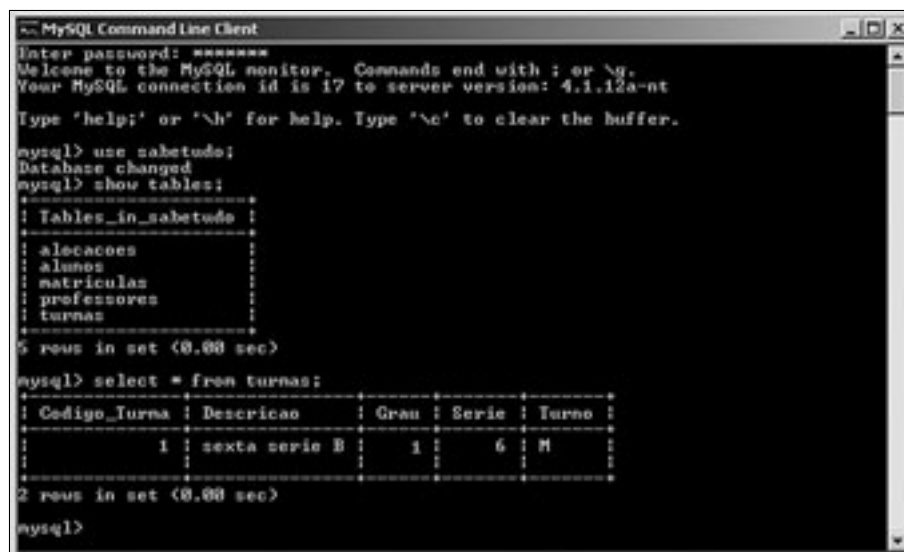
Após executá-la quantas vezes desejar, acesse o banco de dados *SabeTudo*, do MySQL e liste os registros da Tabela de Turmas para ver se batem com as suas operações de inserção.

Caso você não se lembre de como fazer isso, aqui vai algumas dicas:

- Abra o MySQL pelo atalho do programa *MySQL Command Line Client*;
- Na solicitação de senha, informe a palavra **virtual** e pressione a tecla *Enter*;
- Entre no banco de dados *SabeTudo* pelo comando *use sabetudo*;
- Já dentro do DataBase *SabeTudo*, execute o comando *show tables* para verificar as tabelas deste banco de dados;
- Execute o comando *Select * from Turmas* para listar os dados cadastrados na Tabela Turmas.

Agora é só verificar se os dados batem com os valores cadastrados por você.

Na figura abaixo são apresentados os passos descritos anteriormente, porém já executados no MySQL.



```

MySQL Command Line Client
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 17 to server version: 4.1.12a-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use sabetudo;
Database changed
mysql> show tables;
+-----+
| Tables_in_sabetudo |
+-----+
| alecacoes           |
| alunos              |
| matriculas           |
| professores          |
| turnas               |
+-----+
5 rows in set (0.00 sec)

mysql> select * from turnas;
+-----+-----+-----+-----+-----+
|Codigo_Turma | Descricao | Grau | Serie | Turno |
+-----+-----+-----+-----+-----+
|          1  | sexta serie B |    1 |     6 |     M |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>

```

Figura 6.11 - VERIFICAÇÃO DA EXECUÇÃO DA CLASSE USARCOMANDOS

Com isso você tem uma real impressão de como usar a classe ComandosSQL, mas muita coisa ainda pode ser melhorada, principalmente no que se refere à apresentação dos dados a partir de um **Select**, armazenado na classe **ResultSet**, mas isso fica para próxima unidade.

Até breve.



Síntese

Nesta unidade você pôde revisar alguns conceitos da programação em Java, principalmente na estrutura de uma classe, como a mesma pode ser instanciada e usada.

Junto a comandos que você já conhecia e que são essenciais para programação em Java, algumas novas classes foram apresentadas a você, como as classes *DriverManager*, *Connection*, *Statement* e *ResultSet*.

No momento essas estruturas podem parecer complexas, mas lembre-se que se tratam de um assunto novo, recém abordado. À medida que você for interagindo com essas classes, inclusive nas unidades que seguem, o uso se tornará fácil e direto.

Lembre-se que um dia com certeza você se questionou sobre os comandos em Java que hoje você já domina. Naquele momento parecia que você nunca os aprenderia, e hoje são tão banais.

Com as classes de conexão a banco de dados acontecerá o mesmo.



Atividades de auto-avaliação

1. Qual a vantagem em se apresentar na tela, pelo método `getMessage()`, o erro de uma `Exception`, como no exemplo a seguir?

```
try{
    //faz algo;
}catch(Exception e){
    System.out.println("O erro foi : " + e.getMessage());
}
```

[illegible]

2. Na definição de uma classe em Java, é possível atualizar o valor de um atributo diretamente, sem que se use o seu método modificador. Essa forma de implementação pode ser considerada equivocada?



Saiba mais

- DEITEL, H.M;DEITEL,P.J. **Java: como programar**. 6ª ed. Pearson, 2005.
- SANTOS, Rui Rossi dos. **Programando em Java 2- Teorias e Aplicações**. Axcel: 2004.
- SIERRA,Kathy;BATES,Bert. **Java Use a cabeça**. Alta Books: 2005.
- THOMPSON, Marco Aurélio. **Java 2 & Banco de Dados**. São Paulo, Érica: 2002.

Na internet:

<<http://portaljava.com>>
<<http://www.guj.com.br>>

Comandos em Java para apresentação dos dados



Objetivos de aprendizagem

- Compreender a estrutura de retorno de um comando **Select**.
- Compreender os comandos da linguagem Java que manipulam a saída de um comando de seleção.
- Implementar e testar os comandos de retorno de seleção em Java.



Seções de estudo

- Seção 1** Estrutura de um Comando de Seleção.
- Seção 2** Classe ResultSet.
- Seção 3** Classe ResultSetMetaData.
- Seção 4** Passagens de Parâmetros para Seleção.



Para início de conversa

Caro aluno, até aqui você tem criado um conjunto de classes para facilitar a sua vida no que diz respeito à conexão da linguagem Java a um banco de dados. Continuando nesse foco, nesta unidade você criará uma classe para apresentação dos dados retornados por um comando de seleção, por meio de uma tabela.

Com o uso desta tabela a apresentação dos dados se tornará visualmente mais atraente para o usuário da aplicação, principalmente pelo fato da seleção de dados ser uma das atividades mais executadas num sistema com acesso a banco de dados.

SEÇÃO 1 – Estrutura de um comando de seleção

Diferente dos outros comandos em linguagem SQL, o comando de seleção de dados *Select* possui uma característica própria, pois é um comando com retorno de dados.

Quando da execução de um comando *Select*, há um retorno de três estruturas para o usuário do banco de dados, que são:

- o nome das colunas selecionadas pelo comando;
- os dados de cada linha que faz parte do filtro executado pelo comando;
- o tipo de dados de cada coluna.

Por exemplo, no comando: *Select * from turmas*, o retorno do comando será algo como:

```
mysql> select * from turmas;
```

Codigo_Turma	Descricao	Grau	Serie	Turno
1	Sexta Serie C	1	6	M
2	Sexta Serie B	1	6	U
3	Sexta Serie A	1	6	N

3 rows in set (0.00 sec)

Figura 7.1 - RETORNO DO COMANDO SELECT * FROM TURMAS.

No caso do comando anterior, o asterisco (*) faz o papel de especificação dos nomes de todas as colunas da Tabela Turmas, do DataBase SabeTudo.

Acredito que a estrutura do comando *Select* não seja novidade para você, mesmo que você não se lembre de todas as combinações possíveis para esse comando, como o uso de *Where*, *Group By*, *Having* e *Order By*.

Analizando a estrutura representada pela figura anterior, pode-se notar que o tipo de dado de cada coluna não pode ser identificado por meio do retorno do comando *Select*, uma vez que esse comando retorna os dados cadastrados, mas não retorna a estrutura da tabela.

Para verificar o tipo de dado de cada coluna, usa-se o comando *Describe*, conforme a figura a seguir.

```
mysql> describe turmas;
```

Field	Type	Null	Key	Default	Extra
Codigo_Turma	int(4)		PRI	0	
Descricao	varchar(30)				
Grau	int(4)			0	
Serie	int(4)			0	
Turno	char(1)				

Figura 7.2 - RESULTADO DO COMANDO DESCRIBE TURMAS;

Sendo assim, o programador que utiliza o banco de dados precisa saber o modelo utilizado para definição de cada tabela, as regras, os nomes e tipos de cada coluna das tabelas. Por isso, é imprescindível que esse profissional tenha um modelo completo em sua mesa, ou em qualquer lugar de fácil visualização e análise.

Mas como permitir que esses dados de retorno do comando *Select* sejam interpretados pela linguagem de programação Java, uma vez que são estruturas diferentes?

Aqui que entra o ponto fundamental da linguagem de programação e da necessidade de um profissional especializado em banco de dados, pois é preciso que ele tenha fundamentos que lhe permitam compreender o que um comando *Select* retorna.

Sabendo como o comando de seleção funciona, cabe a ele estudar a linguagem a fim de encontrar a estrutura de dados que lhe permita interagir com o banco de dados.

No caso do Java, esse retorno de um comando de seleção é representado por duas estruturas, ou no caso do Java, classes: **ResultSet**; e **ResultSetMetaData**;

Na próxima seção você verá o que faz cada uma dessas classes.

SEÇÃO 2 – Classe *ResultSet*

A classe **ResultSet** como o próprio nome já diz, faz uma referência aos dados retornados por um comando de seleção, ou simplesmente, *Select*.



É importante salientar que essa classe não é uma classe de conexão ao banco de dados, mas sim uma classe de manipulação dos resultados obtidos por um comando de seleção.

O **ResultSet** é a classe responsável por manipular os dados obtidos da execução de uma *query* (pesquisa pelo uso de um *Select*).

Sendo assim, para que a classe **ResultSet** possa ser utilizada é necessário que o banco de dados já tenha sido conectado e aberto.

Até a versão do JDBC 1.0 o **ResultSet** não permitia a navegabilidade dentro dos resultados trazidos. Dessa forma, o comando só podia ser utilizado de maneira seqüencial, sempre para frente. Uma vez que um registro fosse acessado, não era possível voltar para acessá-lo novamente. A versão JDBC 2.0, mais atual, introduziu a **navegabilidade**, o **posicionamento** e a **manipulação de dados** no **ResultSet**.



A **navegabilidade** refere-se à capacidade de navegar para frente e para trás nas linhas contidas em um **ResultSet**. O **posicionamento** é a capacidade de mover o cursor corrente para uma outra posição dentro do **ResultSet**. A **manipulação** é a capacidade de alterar os valores contidos em um **ResultSet**.

Com isso, a classe passou a disponibilizar um conjunto maior de métodos de trabalho, como:

Quadro 7.1 - MÉTODO DA CLASSE *RESULTSET*

Método da Classe <i>ResultSet</i>
first() : posiciona o cursor na primeira linha do <i>ResultSet</i> e retorna true , caso não exista linhas, retorna false.
last() : posiciona o cursor na última linha do <i>ResultSet</i> e retorna true , caso não exista linhas, retorna false.
next() : posiciona o cursor na próxima linha do <i>ResultSet</i> e retorna true , caso a linha exista e false caso não exista mais linhas no <i>ResultSet</i> .
isFirst() : retorna true caso o cursor corrente esteja posicionado na primeira linha.
isLast() : retorna true caso o cursor corrente esteja posicionado na última linha.
isBeforeFirst() : retorna true caso o cursor corrente esteja posicionado antes da primeira linha.
isAfterLast() : retorna true caso o cursor corrente esteja posicionado após a última linha.
int getRow() : retorna o número da linha do cursor corrente.

Como o Java é uma linguagem *Case Sensitive*, em que comandos maiúsculos e minúsculos são diferenciados, no quadro apresentado a sintaxe dos comandos já está na forma correta de uso.

Para compreender melhor o que faz cada um dos métodos apresentados no quadro, abra o JCreator e crie um novo arquivo do tipo *Java File*, renomeie o arquivo para **Exemplos**, salvando-o na pasta **c:\BD2\Fontes**.

Nesta nova classe serão implementadas as seguintes funcionalidades:

- conexão com banco de dados;
- execução de um comando de **Seleção** na Tabela Turmas;
- listagem de todos os dados da Tabela de Turmas;
- navegação na Tabela de Turmas;
- apresentação de um menu de interação com usuário.

Essa classe será manipulada pelo usuário por meio de um Menu com o seguinte *layout*:



Figura 7.3 - MENU DE INTERAÇÃO COM A CLASSE EXEMPLOS

Por meio desse menu o usuário poderá listar todos os dados da Tabela de Turmas, bem como navegar pelos registros cadastrados. Veja a seguir o código-fonte dessa classe:

```
import java.sql.*;
import javax.swing.*;

class Teste {

    private static Connection con;
    private static Statement st;
    private static ResultSet rs;

    public static String Turno(String _Turno){
        if(!_Turno.equalsIgnoreCase("M")) return "Matutino";
        else
            if(!_Turno.equalsIgnoreCase("V")) return "Vespertino";
        else
            return "Noturno";
    }

    public static void Conectar(){
        try{
            String url = "jdbc:odbc:AcessoDB";
            String usuario = "root";
            String senha = "virtual";
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            con = DriverManager.getConnection(url,usuario,senha);
            System.out.println("Conectando.");
        }
    }
}
```

```

    }catch(Exception e){
        System.out.println("Erro de Conexão:" + e.getMessage());
    }
}

public static void Desconectar(){
    try{
        if(st != null){
            st.close();
            con.close();
        }
        System.out.println("Desconectando.");
    }catch(Exception e){
        System.out.println("Erro desconectando o banco.");
    }
}

public static void Selecionar(String Comando){
    try{
        st = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                   ResultSet.CONCUR_READ_ONLY);
        rs = st.executeQuery(Comando);
    }catch(Exception e){
        System.out.println("Comando não pode ser executado!");
    }
}

public static void Listar(){
    try{
        System.out.println("Codigo: " + rs.getString("Codigo_Turma"));
        System.out.println("Turma: " + rs.getString("Descricao") );
        System.out.println("Serie: " + rs.getString("Serie") );
        System.out.println("Grau: " + rs.getString("Grau") );
        System.out.println("Turno: " + Turno(rs.getString("Turno"))+"\n");
    }catch(Exception e){
        System.out.println("Erro listando os dados.");
    }
}

public static void Primeiro() throws SQLException {rs.first();}
public static void Ultimo() throws SQLException {rs.last();}
public static void Proximo() throws SQLException {rs.next();}
public static void Anterior() throws SQLException {rs.previous();}
public static boolean NoPrimeiro() throws SQLException {return rs.isFirst();}
public static boolean NoUltimo() throws SQLException {return rs.isLast();}

public static void ListarTodos() throws SQLException{
    Primeiro();
    do{
        Listar();
        Proximo();
    }while (!rs.isAfterLast());
}

public static void Menu(){
    try{
        Conectar();
        Selecionar("Select * from Turmas");
        int op = 0;
        do{

```

SEGUE ►

```

        op = Integer.parseInt(JOptionPane.showInputDialog(
            "1-Listar Todos\n"+
            "2-Listar o Primeiro Registro\n"+
            "3-Listar o Último Registro\n"+
            "4-Listar o Próximo Registro\n"+
            "5-Listar o Registro Anterior\n"+
            "6-Sair"));
        switch (op){
            case 1: System.out.println("Listando as Turmas ");
                    ListarTodos();
                    System.out.println("-----");
                    break;
            case 2: System.out.println(" --- PRIMEIRO --- ");
                    Primeiro();
                    Listar();
                    System.out.println("-----");
                    break;
            case 3: System.out.println(" --- ULTIMO --- ");
                    Ultimo();
                    Listar();
                    System.out.println("-----");
                    break;
            case 4: if (!NoUltimo()){
                        Proximo();
                        Listar();
                    }else{
                        JOptionPane.showMessageDialog(null,
                            "Não há próximo!");
                    }
                    break;
            case 5: if (!NoPrimeiro()){
                        Anterior();
                        Listar();
                    }else{
                        JOptionPane.showMessageDialog(null,
                            "Não há anterior!");
                    }
                    break;
        }
        //final so switch

        }while (op != 6); //final do do
        Desconectar();
        System.exit(0);
    }catch(Exception e){
        System.out.println("Erro:" + e.getMessage());
    }
}

//final da classe Menu

public static void main (String args[]) {
    Menu();
}

//final da classe Exemplos

```

Veja a seguir o que faz cada uma dos métodos da classe Exemplos.

Quadro 7.2 - MÉTODOS DA CLASSE EXEMPLOS

Método	Objetivo
<code>public static String Turno(String _Turno)</code>	Este método tem como finalidade converter os valores do campo Turno , de char para String . Assim, ao invés de apresentar valores como 'M', 'V' e 'N', as saídas serão, respectivamente, "Matutino", "Vespertino" e "Noturno".
<code>public static void Conectar()</code>	Este método é responsável pela conexão ao Banco de Dados MYSQL, no database "Sabefudo", pela fonte de dados "AcessoDB".
<code>public static void Desconectar()</code>	Esse método verifica se a conexão foi realizada anteriormente, pelas classes Statement (st) e Connection (con). Em seguida fecha a conexão com banco de dados pelo método close() de cada classe.
<code>public static void Selecionar(String Comando)</code>	Esse método prepara o banco de dados para execução de um comando SQL, pela classe do tipo Statement. Na classe ResultSet é armazenado o resultado do comando passado como parâmetro. <i>ResultSet.TYPE_SCROLL_SENSITIVE</i> permite a navegação no banco de dados, pelas funções first(), previous(), last(), etc. <i>ResultSet.CONCUR_READ_ONLY</i> não permite que valores resultantes do comando de seleção sejam alterados.
<code>public static void Listar()</code>	Esse método lista na tela os valores de todos os campos da tabela Turma. O método <i>rs.getString()</i> recebe como parâmetro o nome do campo que será lido, e retorna o valor cadastrado no banco de dados no formato de uma String.
<code>public static void Primeiro()</code> <code>public static void Ultimo()</code> <code>public static void Proximo()</code> <code>public static void Anterior()</code>	Esses métodos permitem a navegação no resultado do comando de seleção que foi executado.
<code>public static void ListarTodos()</code>	Esse método lista todos os registros resultantes do comando de seleção. Para listar todos os dados a lógica adota foi, posicionar no primeiro registro do resultado do comando de seleção. Enquanto não chegar ao final de todos os registros (! <i>rs.isAfterLast()</i>), mostra os dados de cada campo (Listar()) e vai para o próximo (Proximo()) registro.

SEGUIE ►

<code>public static boolean NoPrimeiro()</code> <code>public static boolean NoUltimo()</code>	Esses métodos retornam verdadeiro ou falso para indicar se a posição atual de leitura é a primeira ou a última de todos os registros retornados pelo comando de seleção.
<code>public static void Menu()</code>	Esse método desenha um Menu na tela, sendo que cada opção do menu faz uma chamada aos métodos apresentados acima.

O programa principal da classe, o método *main* (*String args []*), faz uma única referencia ao método *Menu()*, que por sua vez, faz a chamada a todas as funcionalidades do sistema.

Agora que você já está por dentro do que deve fazer a classe acima, edite, compile e execute o código apresentado anteriormente e divirta-se com o funcionamento do programa.

Não se esqueça, em caso de erro de compilação, revise o que você editou e compare com o que está descrito aqui e, principalmente, tenha paciência, pois essa é uma virtude de qualquer programador.

SEÇÃO 3 – Classe ResultSetMetaData

A classe **ResultSetMetaData** faz parte de um conjunto de classes do Java que permitem a análise dos metadados do banco de dados utilizado.

Apesar do nome estranho, **metadados**, o seu significado é bem simples.



Os **metadados** são as informações sobre as tabelas do banco de dados, como nomes das colunas, tipos, tamanho das colunas, usuário que está acessando a tabela.

O principal objetivo do acesso aos dados armazenados no metadados é a independência do código implementado das estruturas das tabelas acessada pelo sistema. Por exemplo, na classe da seção anterior, para listar os dados das tabelas o código usado foi:

```
public static void Listar(){
    try{
        System.out.println("Codigo: "+ rs.getString("Codigo_Turma"));
        System.out.println("Turma: "+ rs.getString("Descricao" ) );
        System.out.println("Serie: "+ rs.getString("Serie" ) );
        System.out.println("Grau: "+ rs.getString("Grau" ) );
        System.out.println("Turno: "+ Turno(rs.getString("Turno"))+"\n");
    }catch(Exception e){
        System.out.println("Erro listando os dados.");
    }
}
```

Note que a solução está presa à estrutura da Tabela Turmas, pois os nomes dos campos são fixos, sendo assim, se este método for executado para outra tabela, não funcionará. Ou seja, em muitas situações se torna necessário recuperar esses metadados para que se construam consultas dinamicamente, pois em uma grande base de dados mudanças estruturais acontecem com certa frequência.

Existem duas classes que permitem recuperar metadados a partir do Driver *JDBC*, no pacote *java.sql*:

- DatabaseMetaData, e
- ResultSetMetaData.

Na versão JDBC 3.0, há também a classe ParameterMetaData, mas não será tratada nesta seção.

A classe DatabaseMetaData

A classe **DatabaseMetaData** é uma classe mais genérica que permite acessar informações relacionadas aos metadados sobre o banco de dados em uso, como:

- objetos do banco de dados;
- informações sobre o *Driver* JDBC ;
- privilégios de acesso.

Essa classe está diretamente relacionada à classe *Conenction*. O código fonte a seguir apresenta um exemplo de uso da classe **DatabaseMetaData**:

```
import java.sql.*;
import javax.swing.*;

public class Metadados {
    public static void main(String args[]) {
        try{
            String url = "jdbc:odbc:AcessoDB";
            String usuario = "root";
            String senha = "virtual";
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection (url,usuario,senha);
            System.out.println("Conectando.");
            DatabaseMetaData dbmd = con.getMetaData();
            JOptionPane.showMessageDialog(null,"Versao do Driver JDBC = "+
dbmd.getDriverVersion()+"\n"Versao do Banco de Dados = "+
dbmd.getDatabaseProductVersion();
        }catch(Exception e){
            System.out.println("Erro de Conexão:"+e.getMessage());
        }
    }
}
```


A classe `DataBase metaData` apresentará na tela a versão do *Driver* JDBC e a versão do banco de dados utilizado. Veja o resultado da execução da classe:

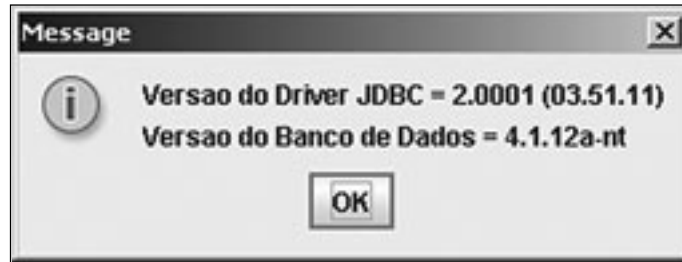


Figura 7.4 - RESULTADO DA EXECUÇÃO DA CLASSE METADADOS

A classe `ResultSetMetaData`

A classe **`ResultSetMetaData`** está relacionada aos metadados das tabelas do banco de dados. Essa classe possibilita a recuperação dinâmica de dados como:

- quantidade de colunas de uma tabela;
- nome da coluna;
- tipo de dados da coluna;
- tamanho de cada coluna.

No código-fonte a seguir é apresentada uma classe que permite resgatar alguns dados referentes à Tabela Turmas.

```
import java.sql.*;
import javax.swing.*;

public class MetadadosTabela {
    public static void main(String[] args) {
        try{
            String url = "jdbc:odbc:AcessoDB";
            String usuario = "root";
            String senha = "virtual";
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection(url,usuario,senha);
            System.out.println("Conectando.");
```

SEGUE ►

```

        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery("SELECT * from Turmas");
        ResultSetMetaData rsmd = rs.getMetaData();
        int colunas = rsmd.getColumnCount();
        String saida = "";
        for (int i=0; i<colunas; i++) {
            saida+="\nColuna:" + rsmd.getColumnName (i + 1) +
                "\nTipo:" + rsmd.getColumnTypeName (i + 1) +
                "\nTamanho:" + rsmd.getColumnDisplaySize (i + 1) + "\n";
        }
        JOptionPane.showMessageDialog(null,saida);

    }catch(Exception e){
        System.out.println("Erro de Conexão:"+e.getMessage());
    }
}
}

```

O resultado da execução da classe será algo como:

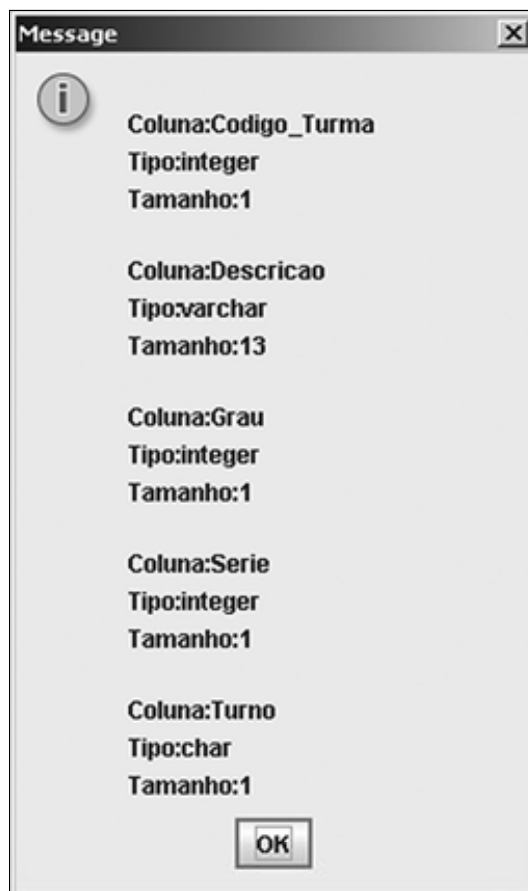


Figura 7.5 - ESTRUTURA DA TABELA TURMAS, LISTADA PELA CLASSE METADADOSTABELA

O ideal é que você implemente essa classe no JCreator, compile e execute para vê-la funcionando. É muito importante que você interaja com essas classes no JCreator, para se habituar com os comandos e praticá-los.

SEÇÃO 4 – Passagens de parâmetros para seleção

Acredito que você tenha notado nas implementações das classes anteriores que uma das principais características é torná-las **genéricas**, de forma que possam atender as mais variadas funcionalidades, sem que o programador tenha que reescrevê-las.



Uma boa metodologia adotada para criação de classes genéricas é o uso de **passagens de parâmetros para o código SQL**.

Por exemplo, para consultar as turmas cadastradas pelo campo chamado “codigo_turma”, o comando SQL ficaria assim:

```
Select * from Turmas where Codigo_Turma = <um_valor>;
```



O que mudaria nesse código para a consulta de várias turmas?

O único item nesse comando que pode ser alterado é o valor do código da turma buscado. Então, porque exigir do programador que ele altere esse comando toda vez que for pesquisar uma nova turma? Seria um desperdício de tempo.

Uma forma de resolver esse problema é a utilização do método *prepareStatement* antes da execução do comando SQL.

Sendo que o valor do campo que pode ser alterado passa a ser representado por um símbolo de interrogação (?).

Veja a classe a seguir que implementa essa regra e depois volto a detalhar como essa técnica funciona:

```
import java.sql.*;

public class usodeParametros {
    public static void main(String[] args) {
        try{
            String url = "jdbc:odbc:AcessoDB";
            String usuario = "root";
            String senha = "virtual";
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection(url,usuario,senha);
            System.out.println("Conectando.");

            PreparedStatement st = con.prepareStatement("SELECT * " +
                "FROM turmas " +
                "WHERE Codigo_Turma = ?");

            st.setString(1,"3");

            ResultSet rs = st.executeQuery();
            while (rs.next()) {
                System.out.println(rs.getString("Codigo_Turma"));
                System.out.println(rs.getString("Descricao"));
            }
            rs.close();
            st.close();
        }catch(Exception e){
            System.out.println("Erro de Conexão:" + e.getMessage());
        }
    }
}
```

O diferencial da classe acima está nos códigos:

```
PreparedStatement st = con.prepareStatement("SELECT *" +  
"FROM turmas" +  
"WHERE Codigo_Turma = ?");
```

O símbolo de interrogação será substituído pelo valor informado na passagem de parâmetros `st.setString(1,"3")`;

O primeiro item do método `st.setString()` representa o número do parâmetro, nesse caso o código tem apenas um parâmetro (ou seja, um ?). O segundo item representa o valor que será buscado na tabela.

Veja a figura a seguir.

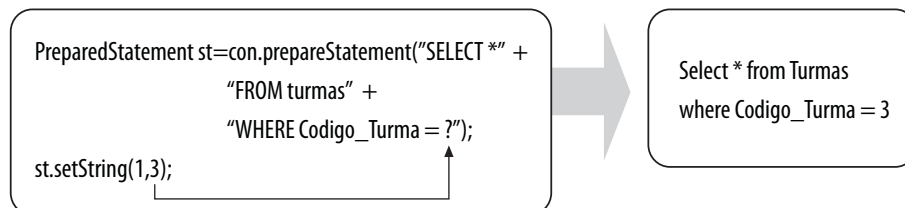


Figura 7.6 - REPRESENTAÇÃO DO COMANDO SELECT APÓS RECEBER O PARÂMETRO

Note que o comando SQL de seleção receberá o valor passado como parâmetro. Podem ser usados quantos parâmetros forem necessários. Veja mais este exemplo, só que agora usando **dois parâmetros**:

```
import java.sql.*;  
  
public class Metadados2 {  
    public static void main(String[] args) {  
        try{  
            String url = "jdbc:odbc:AcessoDB";  
            String usuario = "root";  
            String senha = "virtual";  
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
            Connection con = DriverManager.getConnection(url,usuario,senha);  
            System.out.println("Conectando.");
```

SEGUE ►

```
PreparedStatement st = con.prepareStatement("SELECT * " +
    "FROM turmas " +
    "WHERE Codigo_Turma = ? AND "+
    "Descricao = ?");

st.setString(1,"3");
st.setString(2,"Sexta Serie A");

ResultSet rs = st.executeQuery();
while (rs.next()) {
    System.out.println(rs.getString("Codigo_Turma"));
    System.out.println(rs.getString("Descricao"));
}
rs.close();
st.close();
} catch (Exception e){
    System.out.println("Erro de Conexão:" + e.getMessage());
}
}
```



O uso de parâmetros pode facilitar em muito a vida do programador, pois são muitas as regras de negócio que envolve cada tabela de um banco de dados, regras de inserção, consulta, exclusão e alteração.

No caso da linguagem Java, cada uma dessas tabelas é gerenciada por uma classe de controle, que implementa o que pode e o que não se pode fazer com a tabela por parte do usuário do sistema. Essas classes abrangem todos os códigos vistos por você até agora, sempre buscando a generalidade da classe.

Mas isto é assunto para a próxima unidade.



Síntese

Você pôde estudar nesta unidade que a classe `DatabaseMetaData` é obtida pela instância da classe `Connection` e oferece um conjunto de recursos de recuperação de metadados da base de dados e da sua conexão atual, podendo ser utilizada, por exemplo, para verificar, a versão do **Driver JDBC**.

Já, a classe `ResultSetMetaData` pode ser obtida por um `ResultSet` previamente criado, possibilitando a recuperação de metadados sobre as colunas das tabelas, como: **quantidade de colunas**, **nome** e **tipos** das mesmas.

Com essas duas classes é possível recuperar informações dinamicamente de uma base de dados, tornando as classes implementadas mais abstratas.



Atividades de auto-avaliação

1. Cite exemplos de metadados em banco de dados relacional.

2. De que forma a linguagem Java obtém informações sobre os metadados de um SGDB?



Saiba mais

- ANSELMO, Fernando. **Aplicando orientação a objetos em Java**. Visual Books, 2005.
- COSTA, Luis Carlos da. **Java para iniciantes**. Ciência Moderna, 2002.
- HORSTMANN, Cay S. **Big Java**. Artmed Bookman, 2004.
- RANGEL, Alexandre. **Mysql-Projeto Modelagem e Desenvolvimento de Banco de Dados**. Alta Books, 2005.

Criando a aplicação de cadastro de alunos



Objetivos de aprendizagem

- Aplicar as técnicas desenvolvidas nas unidades anteriores.
- Implementar e testar os processos de cadastramento e povoamento dos dados da Tabela de Alunos.



Seções de estudo

- Seção 1** Estrutura da classe de cadastro de alunos.
- Seção 2** Atributos da classe Alunos.
- Seção 3** Construtor da classe Alunos.
- Seção 4** Verificando se o código de matrícula já existe.
- Seção 5** Preparando os valores para gravação.
- Seção 6** Gravando os dados na Tabela de Alunos.
- Seção 7** Excluindo os dados da Tabela de Alunos.
- Seção 8** Alterando os dados da Tabela de Alunos.



Para início de conversa

Bem, até esta unidade você teve um papel fundamental no processo de desenvolvimento de um sistema computacional com acesso a banco de dados, planejando, analisando e criando mecanismos para se chegar à solução esperada.

A partir de agora, é a hora de colocar isso em prática.

Sendo assim, nesta unidade você atuará como um desenvolvedor de soluções, criando uma classe em Java que permita o cadastramento e a manipulação dos dados dos alunos registrados no banco de dados da escola “Sabe Tudo”.

Chegou a hora de colher os frutos daquilo que você plantou com tanta dedicação, estudo e paciência.

Portanto, mãos à obra.

Seção 1 – Estrutura da classe de cadastro de alunos

Como já visto por você, um programa em Java é representado por meio de uma classe, com seus atributos e métodos.

Sendo assim, a implementação do programa de manipulação da Tabela Alunos, do modelo de dados implementado em MySQL, também será uma classe, que seguirá as regras de negócio do sistema proposto à escola “Sabe Tudo”.

Mas quais são estas regras?

O cadastro de alunos seguirá as seguintes regras:

- a inserção de todos os dados obrigatórios. Caso essa regra seja quebrada, uma mensagem será informada ao usuário do sistema, solicitando novamente os dados para esse campo;
- os campos obrigatórios serão identificados pelo símbolo de asterisco (*);

- antes da gravação dos dados na tabela, o usuário visualizará as informações fornecidas por ele, e o mesmo poderá ou não confirmar o cadastro;
- o usuário não poderá cadastrar um novo aluno com um código de matrícula que já tenha sido usado, caso isso aconteça, uma mensagem será enviada identificando o problema ocorrido;
- a exclusão de dados será realizada com base no código de matrícula informado. Caso a matrícula informada exista no banco de dados, serão apresentados a **matrícula** e o **nome do aluno** selecionado;
- o usuário poderá confirmar ou não a exclusão solicitada;
- a atualização dos dados será realizada a partir da matrícula do aluno. Caso a matrícula seja localizada na base de dados, serão apresentados os dados cadastrados atualmente e serão solicitados os novos dados, para cada campo da tabela;
- a pesquisa de dados do aluno será realizada a partir da **matrícula** ou do **nome do aluno**, sendo que o usuário poderá especificar quais os dados que ele deseja visualizar. O resultado da pesquisa será apresentado no formato de um “*grid*”, para facilitar a interação com usuário.

Bem, com certeza você deve ter imaginado outras regras de negócio para a Tabela de Alunos. No entanto, o objetivo desta unidade é apresentá-lo às técnicas para a realização de um conjunto mínimo de operações, e cabe a você, a partir disso, complementar com as regras que considerar essenciais.

Eu estou apenas plantando a semente, é você que deve regá-la para que dê frutos e cresça a cada dia.

Cada uma das regras acima utilizará um conjunto de comandos em Java aprendidos nesta disciplina nas unidades anteriores, e comandos já estudados por você na disciplina de Programação Orientada a Objetos.

Para facilitar a sua compreensão do código-fonte, serão apresentadas as regras de implementação separadamente, e no final da unidade você terá o mesmo código, porém com todas as regras implementadas em Java.

O objetivo é dividir para conquistar o conhecimento desejado.

Seção 2 – Atributos da classe Alunos

Como a classe que será implementada representa as regras de negócio da Tabela de Alunos, a mesma será nomeada de “Alunos”.

A classe Alunos precisará importar pacotes de classes:

- `javax.swing;`
- `java.sql.`

O pacote *javax.swing* será necessário para as operações de entrada e saída de dados da classe, por meio dos métodos:

- `JOptionPane.showInputDialog;` e
- `JOptionPane.showMessageDialog.`

Já o pacote *java.sql* será necessário para as operações com banco de dados, por meio das classes:

- `ResultSet;`
- `ResultSetMetaData.`

Ambas as classes já foram apresentadas nas unidades anteriores.

Com base nisso, a classe Alunos já terá o seguinte escopo:

```
import javax.swing.*;
import java.sql.*;
class Alunos{
    Regras de Negócio da Classe;
}
```

Bem, agora que você já conhece as necessidades de classes externas para o funcionamento da classe Alunos, o próximo passo é a apresentação dos atributos que definem as suas características.

Lembre-se sempre que os atributos de uma classe são sempre privados, ou do Java, *private*.

Os atributos da classe Alunos são:

```
private String Campos[] = {
    "Matrícula *",
    "Nome *",
    "Data de Nascimento *",
    "Idade *",
    "CPF",
    "RG",
    "Nome do Pai *",
    "Nome da Mãe *",
    "Endereço *",
    "Número *",
    "Cidade *",
    "Estado *",
    "Bairro *",
    "Telefone",
    "Celular",
    "Responsável",
    "Alérgico a",
    "Grupo Sanguíneo",
    "Doador de Órgãos Sim/Não"
};
private ComandosSQL Comando;
private ResultSetMetaData Rsmd;
private String NomesColunas[] = new String [19];
```

Antes que você se assuste com os códigos acima, já vou detalhar qual a finalidade do atributo apresentado anteriormente.

Atributo Campos

O atributo Campos é um vetor de *string*, sem tamanho pré-definido, por isso a sintaxe `String Campos [] = { }`;

Esse atributo representa o **Nome de Apresentação** de cada coluna da Tabela de Alunos, porém num português claro, correto, com acentuações.

Lembre-se que quando você cria as colunas da tabela, por meio do comando **Create Table**, você não as acentua.

Dessa forma, ficaria muito estranho a solicitação ao usuário do sistema como uma mensagem como:

Informe o **Endereco**?

Para resolver esse problema, o atributo Campos representa cada uma das colunas da Tabela de Alunos do banco de dados, seguindo a mesma ordem física dos dados, definida no processo criação da tabela e que pode ser visualizada por meio do comando **Describe do SQL**.

Sendo assim, de forma resumida, pode-se dizer que o atributo Campos representa a saída do comando em SQL, *Describe Alunos*, como mostra a tabela a seguir:

Nome Físico	Nome de Apresentação
Matricula	Matrícula *
Nome	Nome *
Data_Nascimento	Data de Nascimento *
Idade	Idade *
CPF	CPF
RG_Identidade	RG
Nome_Pai	Nome do pai *
Nome_Mãe	Nome da Mãe *
Endereco	Endereço *
Numero	Número *
Cidade	Cidade *
Estado	Estado *
Bairro	Bairro *
Telefone	Telefone
Celular	Celular
ChamarPor	Responsável
Alergia	Alérgico à
GrupoSangue	Grupo Sanguíneo
Doador	Doador de Órgãos Sim/Não

Com base nesse vetor do tipo *string*, com 19 posições, iniciando em 0 e terminando no índice 18, é possível representar o nome de todas as colunas da Tabela de Alunos.

Como esse vetor não muda, ou seja, os nomes de todas as colunas são os mesmos sempre, o atributo foi criado como uma lista

constante, em que os valores de cada posição estão representados entre os sinais de abre chaves '{' e fecha chaves '}'.

Para os campos obrigatórios da Tabela de Alunos, foi acrescentado um sinal de asterisco "*" ao lado do nome de apresentação da coluna para que o usuário possa identificar aquele dado como obrigatório.

Atributo Comando

O atributo Comando é uma composição (quando uma classe faz referência direta a outra classe) da classe **ComandosSQL**, apresentada e implementada nas unidades anteriores.

A principal finalidade dessa classe é permitir a conexão com banco de dados por meio do método **AbrirConexão** e possibilitar a execução de comandos em SQL por meio do método **ExecutarComando**.

Duas características importantes da classe **ComandosSQL**, e conseqüentemente do atributo Comando, são os métodos recuperadores **getConectou()** que indica se a conexão com banco de dados foi realizada ou não, e o método **getResultadoSQL()** que representa o resultado da execução de um comando de *Select*.

Atributo Rsmd

Esse atributo com nome estranho é a abreviatura do tipo de classe que ele representa, que é uma classe chamada **ResultSetMetaData**, que também foi vista por você nas unidades anteriores.

É utilizado para:

- identificação do número de colunas da tabela – *getColumnCount()*;
- identificação dos nomes físicos de cada coluna da tabela – *getColumnName()*;
- identificação do tipo de dado de cada coluna da tabela – *getColumnType()*;

Essas identificações são fundamentais para a geração do comando de inserção de dados na Tabela de Alunos, pois o comando ***Insert*** deve referenciar os nomes físicos de cada coluna, e os campos do tipo ***Varchar***, ***Char*** e ***Date*** devem ter o valor de cadastro entre aspas.

Por exemplo, no caso do cadastro da **Matrícula** e do **Nome do Aluno**, o comando ***Insert*** deverá ter a seguinte sintaxe:

```
Insert into Alunos (Matricula, Nome) values (1, "Marcelo Medeiros");
```

Os campos "**Matricula**" e "**Nome**" representam as colunas físicas da Tabela de Alunos, já os valores de cadastros são informado pelo usuário do sistema, entretanto, os dados literais, como **Nome**, **devem estar entre aspas**.

Essas regras de recuperar o nome físico dos campos e acrescentar aspas aos campos literais e data devem ser geradas de forma automática pelo sistema, sem a interação do usuário.

Atributo NomesColunas

O atributo NomesColunas é um vetor de *string*, de tamanho 19, que armazenará os nomes físicos das colunas da Tabela de Alunos.

Esse atributo está diretamente relacionado ao atributo Rsmd, visto anteriormente, uma vez que a geração dos nomes físicos das colunas é realizada por meio da classe ResultSetMetaData.

Basicamente, a sua finalidade é armazenar os nomes das colunas retornadas pela classe ResultSetMetaData, por meio do atributo Rsmd e do método getColumnNames().

Seção 3 – Construtor da classe Alunos

Como já visto por você, uma classe representa as regras de manipulação de um objeto, e para ser utilizada precisa ser instanciada – criada.

O processo de criação da classe é realizado pelo método chamado de **construtor**, que deve ter o mesmo nome da classe implementada.

O principal objetivo do método construtor é definir um valor inicial para os atributos da classe.

Com base nisso, o construtor da classe Alunos tem o seguinte código-fonte:

1	public Alunos(){
2	Comando = new ComandosSQL();
3	Comando.AbrirConexao("AcessoDB","root","virtual");
4	if (!Comando.getConectou()){
5	JOptionPane.showMessageDialog(
6	null,"Erro de Conexão com Banco de Dados\n" +
7	" Entre em contato com Suporte ! \n" +
8	" A aplicação será FECHADA ! ");
9	System.exit(0);
10	}
11	else{
12	try{
13	Comando.ExecutarComando("select * from alunos");
14	ResultSet res = Comando.getResultadoSQL();
15	GerarNomesColunas(res);
16	}catch(Exception e){
17	System.out.println("Erro Gerando Nome das Colunas");
18	}
19	}
20	}

Vou apresentar a você qual a finalidade das principais linhas de comando do código-fonte acima.

Linha 1

Representa o nome do método **construtor**, que deve ter o mesmo nome da **classe**. Tenha sempre o cuidado com maiúsculas e minúsculas, pois a linguagem Java é chamada de *Case Sensitive*, sendo assim, diferencia maiúsculas e minúsculas.

Linha 2

Instancia a classe `ComandosSQL`, representada pelo atributo `comandos`. A criação da classe é realizada pela instrução *new* (novo).

Uma classe só pode ser utilizada se anteriormente ela tenha sido instanciada, caso contrário, será gerada um erro de *NullPointerException*.

Linha 3

Após a criação da classe `ComandosSQL`, é realizada a tentativa de conexão com o banco de dados pelo método **AbrirConexão**, que recebe como parâmetros o nome da fonte (criada via ODBC), o nome do usuário administrado do banco (*root*) e a senha de acesso (**virtual**).

Linha 4

Nessa linha de comando é verificado se a conexão com banco de dados foi realizada (`getConectou()` retornando **true**). Caso a conexão não tenha sido realizada, uma mensagem será apresentada ao usuário e a aplicação será finalizada (`System.exit(0)`), pois sem acesso ao banco de dados, o usuário não poderá realizar nenhuma operação no sistema.

Linha 13

A chegada nessa linha pela execução do programa indica que a conexão foi realizada, sendo assim, é executada uma seleção de todos os registros da Tabela de Alunos para obtenção de um **ResultSet**, representado pelo atributo local **res**.

Linha 15

O método GerarNomesColunas recebe como parâmetro o **ResultSet** do atributo **res** e, pela classe ResultSetMetaData, preenche o atributo **NomesColunas** com o nome físico de cada coluna da Tabela de Alunos.

Veja o código do método **GerarNomesColunas**:

1	public void GerarNomesColunas(ResultSet rs){
2	try{
3	Rsmd = rs.getMetaData();
4	for (int i = 1; i <= Rsmd.getColumnCount(); i++)
5	NomesColunas[i-1] = Rsmd洗getColumnName(i);
6	}catch(Exception e){
7	System.out.println("Erro Gerando Nome das Colunas");
8	}
9	}

Esse método recebe um *ResultSet* por meio do parâmetro **rs**. Esse *ResultSet* é analisado pela classe ResultSetMetaData, representado pelo atributo **Rsmd**.

Um laço **for** é executado, varrendo todas as colunas retornadas pelo comando “**Select * from Alunos**”, e pelo método *getColumnName(i)*, cada uma dessas colunas tem o seu nome físico armazenado no atributo NomeColunas.

Seção 4 – Verificando se o código de matrícula já existe

A Tabela de Alunos possui uma restrição do tipo chave primária que define que não podem existir mais de um aluno com o mesmo número de matrícula.

No caso do Java, caso seja cadastrado mais de um aluno com o mesmo código de matrícula, será gerado um erro, identificado como : **Duplicate entry ... for key**.

Para evitar que esse erro ocorra, que é um erro grave, gerando a interrupção na execução do programa, um método será implementado para verificar se uma determinada matrícula já foi cadastrada.

O método, chamado de **MatriculaExiste**, retornará um valor lógico, verdadeiro ou falso, para indicar se o código já foi ou não utilizado.

Esse método possui o seguinte código-fonte:

1	public boolean MatriculaExiste(String Mat){
2	try{
3	Comando.ExecutarComando(
4	“select * from alunos where Matricula = “ + Mat);
5	ResultSet rs = Comando.getResultadoSQL();
6	return (rs.next());
7	}catch(Exception e){
8	System.out.println(“Erro Pesquisando Matricula.”);
9	return true;
10	}
11	}

Veja o que faz cada linha.

1- O método **MatriculaExiste** recebe um parâmetro do tipo *string*, chamado **Mat**, e retorna um booleano.

3- É realizada uma consulta na Tabela de Alunos, filtrando todos os registros que possuam a matrícula com valor igual ao valor do parâmetro chamado Mat.

5- O atributo local **rs**, do tipo **ResultSet**, armazena o resultado da execução do comando filtro realizado na linha 3 da rotina.

6- O método **next()** retornará **true** (verdadeiro) caso exista pelo menos um registro dentro do *ResultSet*, o que indica que existe algum aluno com a mesma matrícula.

9- Caso, por algum motivo, essa rotina não possa ser executada, o tratamento de exceção do Java retornará um valor verdadeiro, para evitar que sejam cadastradas duas matrículas com mesmo valor, pois a validação não pode ser realizada.

Seção 5 – Preparando os valores para gravação

Como apresentado, os campos gravados na Tabela de Alunos, dependendo do tipo de dado da coluna, podem exigir que o valor seja colocado entre aspas. É o caso dos tipos de dados *Varchar*, *Char* e *Date*.

Para o usuário do sistema isso é abstraído. Quando ele for cadastrar um nome de aluno, ele não fará a entrada de dados inserindo as aspas antes e depois de cada valor.

Por exemplo, para cadastrar o aluno chamado Lucas, ele não precisará informar o dado no formato “Lucas”, quem deverá gerar essa estrutura é o programa, de forma automática.

Para que isso seja realizado, será implementado um método que verificará o tipo da coluna de inserção e, dependendo do tipo, o valor será ou não acrescido de aspas.

O código-fonte do método é:

1	public String FormatoCampo(int Pos,String Valor){
2	String Saida=Valor;
3	try{
4	switch(Rsmd.getColumnType(Pos)) {
5	case Types.VARCHAR: Saida = "" + Valor +"";break;
6	case Types.DATE: Saida = ""+Valor +"";break;
7	case Types.INTEGER: Saida = Valor;break;
8	case Types.CHAR: Saida = ""+Valor+"";break;
9	case Types.FLOAT: Saida = Valor;break;
10	}
11	return Saida;
12	}catch(Exception e) {
13	return Saida;
14	}
15	}

Veja o que faz o código acima.

O método **FormatoCampo** recebe dois parâmetros, o primeiro, chamado de **Pos**, do tipo inteiro, representa qual coluna da tabela será analisada. O segundo parâmetro, chamado de **Valor**, é do tipo *string*, representa o valor que será gravado na Tabela de Alunos.

Esse método retornará o mesmo valor de entrada, com ou sem aspas, dependendo do tipo de dado. Caso o tipo de dado seja *Varchar*, *Char* ou *Date*, o valor de entrada recebe aspas antes e depois do valor, por meio do comando:

$$\text{Saída} = \text{' ' + Valor + ' '};$$

O atributo **Saída** representa o valor de entrada, acrescido ou não das aspas.

Caso a rotina não possa ser executada, o tratamento de exceção do método retornará o mesmo valor de entrada.

Seção 6 – Gravando os dados na Tabela de Alunos

O principal processo da classe Alunos é a rotina de inserção dos dados na Tabela de Alunos.

O processo de inserção deve obedecer às regras de chave primária, respeitar os tipos de dados de cada coluna e a necessidade de cadastro dos campos obrigatórios.

Sendo assim, não é possível que duas matrículas com mesmo valor sejam cadastradas, bem como campos do tipo numeral não poderão ser preenchidos com valores literais e vice-versa.

Basicamente a rotina de inserção de dados, chamada de **CadastrarDados**, é uma rotina de montagem do comando *Insert*.

Um comando *Insert* é formado pelo nome das colunas que serão preenchidas e o valor de cada coluna.

A sintaxe do comando *Insert* é: “**insert into tabela (colunas) values (valores);**”.

As regras adotadas no código de inserção de dados foram:

- solicitar o valor de cada campo da Tabela de Alunos ao usuário do sistema. O nome de cada campo está armazenado no vetor chamado **Campos**;
- armazenar no vetor **Valores** o valor informado pelo usuário, por meio do método JOptionPane.showInputDialog;
- verificar se a matrícula informada já foi cadastrada no sistema;
- verificar se o campo informado corresponde à informação de Doador de Órgãos, caso seja, gravar 1 ou 0 nesse campo;
- verificar quais os campos foram informados pelo usuário e armazenar os nomes físicos de cada coluna e o seu valor de cadastro por meio dos atributos **Colunas** e **Dados**;
- solicitar a confirmação do cadastro.

Veja essas regras codificadas em Java:

1	public void CadastrarDados(){
2	int i=0,itens=0,j=0;
3	String Escolha = "5";
4	String Mensagem = " Confirma o Cadastro dos Seguintes Dados: \n";
5	String Colunas="",Dados="",Operacao="";
6	String ValorLido="";
7	JOptionPane.showMessageDialog(null," Cadastro de ALUNOS \n"+
8	"Os campos com * são Obrigatórios!");
9	String Valores[] = new String[19];
10	
11	while (i < 19){
12	Valores[i] = JOptionPane.showInputDialog(null,Campos[i]);
13	ValorLido = Valores[i];
14	do{
15	if (i == 0 && MatriculaExiste(ValorLido)){
16	JOptionPane.showMessageDialog(null,"Este Número de Matrícula já foi usado!\n"+
17	"Informe um novo código para Matrícula");
18	
19	ValorLido = JOptionPane.showInputDialog(null,Campos[i]);
20	Valores[i] = ValorLido;
21	}

SEGUIE ►

22	}while(i==0 && MatriculaExiste(ValorLido));
23	
24	if(Valores[i].equals("") && (Campos[i].indexOf("*") != -1)){
25	JOptionPane.showMessageDialog(null," Este campo é obrigatório \n"+
26	" Por Favor, recadastre-o!");
27	}
28	else{
29	i++;
30	}
31	if (!ValorLido.equals(""))
32	itens++;
33	}
34	if (!Valores[18].equals("")){
35	if (Valores[18].charAt(0) == 'S' Valores[18].charAt(0) == 's')
36	Valores[18] = "1"; //Sim
37	else
38	Valores[18] = "0"; //Não
39	}
40	j=0;
41	for(i=0;i<19;i++){
42	if (!Valores[i].equals("")){
43	Mensagem = Mensagem + Campos[i] + " : "+ Valores[i]+\n";
44	Colunas=Colunas+NomesColunas[i];
45	Dados =Dados +FormatoCampo(i+1,Valores[i]);
46	if (j < itens-1){
47	Colunas=Colunas + ",";
48	Dados = Dados + ",";
49	
50	}
51	j++;
52	}
53	}
54	Escolha = JOptionPane.showInputDialog(Mensagem + "\n Sim(S) - Não(N)");
55	if (Escolha.equalsIgnoreCase("S")){
56	Operacao = "insert into alunos("+Colunas+") values (" +Dados+");
57	try{
58	Comando.ExecutarComando(Operacao);
59	}catch(Exception e){
60	System.out.println(e.getMessage());
61	}
62	}
63	
64	}

Abaixo um comentário sobre as principais operações realizadas pelo método de cadastro de alunos.

1	Nome do método.
2	Atributos para obtenção do nome dos campos, total de campos cadastrados e obtenção do nome físico das colunas.
3	Atributo para confirmação ou não do cadastro.
4	Atributo que apresentará os dados informados pelo usuário.
5	Atributo para armazenamento dos nomes das colunas, do valor de inserção de cada coluna e armazenamento do comando Insert gerado.
6	Atributo para armazenamento do valor informado pelo usuário.
7	Um aviso para o usuário de que os campos com * são obrigatórios.
9	Atributo para armazenamento de todos os valores informados pelo usuário do sistema.
11	Realiza um laço para a solicitação do valor de cadastro para todos os campos da tabela.
12	Solicita o valor do campo.
14	Repete o cadastro da matrícula até que seja informado um código que ainda não foi usado.
15	Verifica se o valor informado é o da matrícula e se a mesma já foi cadastrada.
24	Verifica se o campo cadastrado é um campo obrigatório, ou seja, possui um asterisco (*).
31	Verifica se o valor do campo cadastrado é diferente de vazio. Incrementa o contador de campos cadastrados.
34	Verifica se o campo informado é o campo de Doação de Órgãos. Se for, troca o valor Sim por 1 e o valor Não por 0 .
41	Verifica os valores cadastrados para todos os campos.
42	Se o valor não for vazio.
43	Monta a mensagem com o nome do campo e o valor cadastrado.
44	Monta o atributo Colunas com os nomes físicos dos campos preenchidos;
45	Monta o atributo Dados com os valores informados para cada campo, verificando se o valor deve ou não ficar entre aspas.
46	Se não for o último campo cadastrado, acrescenta uma vírgula para separar cada nome de coluna e cada valor informado pelo usuário.
54	Apresenta os valores cadastrados pelo usuário e solicita a confirmação do cadastro.
55	Se a escolha for sim, grava os dados na tabela de alunos.

Seção 7 – Excluindo os dados da Tabela de Alunos

Para realizar a exclusão dos dados da Tabela de Alunos é preciso especificar um campo de referência que identifique de forma única o registro do aluno a ser excluído.

Como a Tabela de Alunos possui como chave primária a **matrícula**, a exclusão usará essa informação como referência para identificação e exclusão do aluno.

Porém, antes de excluí-lo, é necessário que se verifique se o aluno pesquisado existe ou não na base de dados.

Com base nessas premissas, o código do método de exclusão de alunos terá a seguinte estrutura:

1	public void ExcluirPorMatricula(){
2	String Escolha = "S";
3	String Saida = "";
4	String Mat = JOptionPane.showInputDialog("Informe a Matrícula para Exclusão:");
5	try{
6	Comando.ExecutarComando("select * from alunos where Matricula = " + Mat);
7	ResultSet rs = Comando.getResultadoSQL();
8	while(rs.next())
9	{
10	Saida = "Matrícula:" + rs.getString("Matricula")+"\n"+
11	"Nome : " + rs.getString("Nome");
12	}
13	if (Saida.equals("")){
14	JOptionPane.showMessageDialog(null,"Matrícula Inexistente!");
15	}
16	else{
17	JOptionPane.showInputDialog("Confirma a exclusão do Aluno:\n"+
18	Saida+"\n"+
19	"Sim(S) - Não(N)");
20	if (Escolha.equalsIgnoreCase("S"))
21	Comando.ExecutarComando(
22	"Delete from alunos where Matricula = " + Mat);
23	}
24	}catch(Exception e){
25	System.out.println("Exclusão não pode ser feita!" + e.getMessage());
26	}
27	}

Para saber como funciona essa implementação, veja a descrição a seguir:

2	Confirma a exclusão.
3	Atributo para apresentação do nome e da matrícula do aluno selecionado.
4	Solicita a matrícula do aluno a ser excluído.
6	Seleciona os alunos com a matrícula informada.
7	Armazena o resultado do Select .
8	Varre todos os registros retornados pelo Select .
11	Armazena o nome e a matrícula do aluno localizado pelo Select .
13	Se o atributo Saida estiver vazio, é porque nenhum aluno foi localizado.
16	Se o atributo Saida é diferente de vazio, então apresenta na tela o nome e a matrícula do aluno e solicita a confirmação da exclusão.
20	Se a escolha for Sim , exclui os alunos com a matrícula informada.

Seção 8 – Alterando os dados da Tabela de Alunos

O processo de alteração de dados do aluno é muito semelhante ao processo de exclusão, pois tanto a operação de exclusão quanto a operação de alteração só podem ser realizadas se o aluno pesquisado existir no banco de dados.

Com relação à alteração, o usuário do sistema poderá visualizar o valor cadastrado para cada campo da tabela e alterá-lo se assim desejar, com exceção da matrícula, pois esse campo é único por se tratar de uma chave primária. O aluno poderá ter qualquer um dos seus dados alterados, menos a matrícula.

O processo de alteração possui as regras de pesquisas já utilizadas no processo de exclusão para saber se o aluno está ou não cadastrado e utiliza também as regras de cadastramentos dos dados para a nova inserção de valores em cada campo da tabela.

Por isso, que ao analisar o código-fonte da rotina de alteração, você encontrará muitas expressões que já são de seu conhecimento.

Veja a seguir a codificação do processo de alteração de dados dos alunos:

1	public void AlterarPorMatricula(){
2	int i;
3	String Valores[] = new String[19];
4	String Saida = "";
5	String Escolha = "S";
6	String ValorLido = "";
7	String Colunas = "";
8	String Dados = "";
9	String Operacao = "";
10	String Mat = JOptionPane.showInputDialog(
11	"Informe a Matrícula para Alteração:");
12	try{
13	Comando.ExecutarComando(
14	"select * from alunos where Matricula = " + Mat);
15	ResultSet rs = Comando.getResultadoSQL();
16	if (rs.next()){
17	Rsmd = rs.getMetaData();
18	for (i=1;i <= Rsmd.getColumnCount();i++){
19	Valores[i-1] = rs.getString(Rsmd.getColumnName(i));

SEGUIE ►

20	if (i == 19){
21	if(Valores[i-1].equals("1"))
22	Valores[i-1] = "Sim";
23	else
24	Valores[i-1] = "Não";
25	}
26	Saida = Saida + Campos[i-1]+":"+Valores[i-1]+"\\n";
27	}
28	}
29	if (Saida.equals("")){
30	JOptionPane.showMessageDialog(null,"Matrícula Inexistente!");
31	}
32	else{
33	Escolha = JOptionPane.showInputDialog("Confirma a alteração do Aluno:\\n"+
34	Saida+"\\n"+
35	"Sim(S) - Não(N)");
36	if (Escolha.equalsIgnoreCase("S")){
37	i=2;
38	while (i<=19){
39	Valores[i-1] = JOptionPane.showInputDialog(
40	Campos[i-1],Valores[i-1]);
41	if(Valores[i-1].equals("") && (Campos[i-1].indexOf("") != -1)){
42	JOptionPane.showMessageDialog(null,
43	" Este campo é obrigatório \\n"+
44	" Por Favor, recadastre-o!");
45	}
46	else{
47	if (i==19 && (!Valores[i-1].equals(""))){
48	if (Valores[i-1].charAt(0) == 'S'
49	Valores[i-1].charAt(0) == 's')
50	Valores[i-1] = "1"; //Sim
51	else
52	Valores[i-1] = "0"; //Não
53	}
54	i++;
55	}
56	}
57	Operacao = "Update Alunos set ";
58	for (i=1;i<19;i++){
59	if (!Valores[i].equals("")){
60	Operacao = Operacao + NomesColunas[i] + "=" +
61	FormatoCampo(i+1,Valores[i]);

SEGUE ►

62	if (i < 18){
63	Operacao = Operacao + ';;
64	}
65	}
66	}
67	Operacao = Operacao + " where Matricula = "+ Mat;
68	Comando.ExecutarComando(Operacao);
69	}
70	}
71	}catch(Exception e){
72	System.out.println("Alteração não pode ser feita!" + e.getMessage());
73	}
74	
75	}

Como essa rotina é um pouco mais extensa que as anteriores, vou descrevê-la por blocos, para facilitar o seu entendimento, ok?

```
public void AlterarPorMatricula(){
    int i;
    String Valores[] = new String[19];
    String Saida = "";
    String Escolha = "S";
    String ValorLido = "";
    String Colunas = "";
    String Dados = "";
    String Operacao = "";
    String Mat = JOptionPane.showInputDialog(
        "Informe a Matrícula para Alteração:"
    );
}
```

Nessas linhas iniciais da rotina de alteração são apresentados os atributos de armazenamento e é realizado o processo de solicitação da matrícula do aluno que terá os dados alterados.

```

Comando.ExecutarComando(
    "select * from alunos where Matricula = " + Mat);
ResultSet rs = Comando.getResultadoSQL();
if (rs.next()){
    Rsmd = rs.getMetaData();
    for (i=1;i <= Rsmd.getColumnCount();i++){
        Valores[i-1] = rs.getString(Rsmd.getColumnName(i));
        if (i == 19){

            if(Valores[i-1].equals("1"))
                Valores[i-1] = "Sim";

            else
                Valores[i-1] = "Não";
        }
        Saida = Saida + Campos[i-1]+":"+Valores[i-1]+"\n";
    }
}

```

Nesse bloco de comandos, o primeiro processo corresponde à execução de um comando **Select** para verificar se a matrícula pesquisada está cadastrada no banco de dados.

A condição **rs.next()** irá retorna **True** (verdadeiro) caso exista um aluno cadastrado com o código de matrícula pesquisado.

Com a utilização da classe `ResultSetMetaData` é possível realizar um laço que passe por todas as colunas da Tabela de Alunos, iniciando na coluna da posição 1 até a coluna final, representada pelo método `getColumnName()`.

Com base na coluna acessada, utilizando-se do método `getString()`, os valores de cada coluna da tabela, ou seja, os dados já cadastrados no banco de dados, são armazenados no vetor chamado **Valores**.

Como o campo da Tabela de Alunos, chamado **Doador**, é um inteiro, ali só pode ser armazenado um valor **1 quando o aluno é doador** e um valor **0 quando ele não é doador**. Para evitar que seja apresentada uma mensagem do tipo, **Doador = 1**, é feita uma conversão de 1 para Sim e 0 para Não.

Todos os dados, como nome do campo e valor do campo, são armazenado no atributo do tipo *string* chamado **Saída**.

```

if (Saida.equals("")){
    JOptionPane.showMessageDialog(null,"Matrícula Inexistente!");
}
else{
    Escolha = JOptionPane.showInputDialog("Confirma a alteração do Aluno:\n"+
        Saida+"\n"+
        "Sim(S) - Não(N)");
    if (Escolha.equalsIgnoreCase("S")){
        i=2;
        while (i<=19){
            Valores[i-1] = JOptionPane.showInputDialog(
                Campos[i-1],Valores[i-1]);
            if(Valores[i-1].equals("") && (Campos[i-1].indexOf("*") != -1)){
                JOptionPane.showMessageDialog(null,
                    " Este campo é obrigatório \n"+

                " Por Favor, recadastre-o!");
            }
            else{
                if (i==19 && (!Valores[i-1].equals(""))){
                    if (Valores[i-1].charAt(0) == 'S' ||
                        Valores[i-1].charAt(0) == 's')
                        Valores[i-1] = "1"; //Sim
                }
                else
                    Valores[i-1] = "0"; //Não
            }
            i++;
        }
    }
}

```

Se o atributo **Saída** estiver com o valor de nulo, vazio, isso significa que a matrícula pesquisada não existe no banco de dados. Caso contrário, será solicitado ao usuário o valor de cada campo da tabela para que ele faça um cadastramento.

No processo de cadastramento, por meio do **JOptionPane.showInputDialog**, o usuário verifica o nome do campo pelo atributo **Campos[i-1]** e visualiza o valor atual cadastrado, pelo do atributo **Valores[i-1]**. Caso o valor não precise ser alterado, basta que ele clique no botão **Ok**, na janela que se abriu.

Nos campos obrigatórios, será necessário que o usuário informe um valor, caso contrário o sistema mandará uma mensagem e só continuará o cadastro quando a regra de obrigatoriedade for obedecida.

Para o campo Doador, será feita uma conversão de Sim para 1 e Não para 2.

```

Operacao = "Update Alunos set ";
for (i=1;i<19;i++){
    if (!Valores[i].equals("")){
        Operacao = Operacao + NomesColunas[i] + "=" +
        FormatoCampo(i+1,Valores[i]);
        if (i < 18){
            Operacao = Operacao + ",";
        }
    }
}
Operacao = Operacao + " where Matricula = " + Mat;
Comando.ExecutarComando(Operacao);
}
} catch (Exception e){
    System.out.println("Alteração não pode ser feita!" + e.getMessage());
}
}

```

Nesse bloco final é gerada a sintaxe do comando em SQL para alteração dos dados da tabela, de forma que o atributo **Operação** tenha um valor parecido como:

```

Update Alunos set Campo1 = Valor1, Campo2 = Valor2, ..., CampoN=Valor N
where Matricula = Valor_Procurado.

```

Note que no último campo alterado ele não é separado por vírgula, por isso a condição **if(i < 18)** no código para verificar se a rotina está montando a sintaxe para o último campo alterado da tabela.

Com mais esta seção, você pode verificar todo o processo de cadastramento e povoamento de dados da Tabela de Alunos.

Mas ainda falta o processo de pesquisa de dados dos alunos, na qual o usuário informa um nome ou uma matrícula e o sistema apresenta os dados cadastrados.

Mas isso é assunto para próxima unidade.

Por enquanto, dê uma analisada no código a seguir que apresenta todas as rotinas implementadas pela classe *Alunos*:

1	import javax.swing.*;
2	import java.sql.*;
3	import java.util.*;
4	
5	class Alunos{
6	
7	private ResultSetMetaData Rsmd;
8	private ComandosSQL Comando;
9	private String NomesColunas[] = new String [19];
10	private String Campos[] =
11	{“Matrícula * “,
12	“Nome * “,
13	“Data de Nascimento * “,
14	“Idade * “,
15	“CPF”,
16	“RG”,
17	“Nome do Pai * “,
18	“Nome da Mãe * “,
19	“Endereço * “,
20	“Número * “,
21	“Cidade * “,
22	“Estado * “,
23	“Bairro * “,
24	“Telefone “,
25	“Celular”,
26	“Responsável”,
27	“Alérgico a”,
28	“Grupo Sanguíneo”,
29	“Doador de Órgãos Sim/Não”
30	};
31	
32	
33	public Alunos(){
34	Comando = new ComandosSQL();
35	Comando.AbrirConexao(“AcessoDB”,“root”,“virtual”);
36	if (!Comando.getConectou()){
37	JOptionPane.showMessageDialog(null,“Erro de Conexão com Banco de Dados\n” +
38	“ Entre em contato com Suporte ! \n” +
39	“ A aplicação será FECHADA ! “);
40	System.exit(0);
41	}
42	else{

SEGUIE ►

43	try{
44	Comando.ExecutarComando("select * from alunos");
45	ResultSet res = Comando.getResultadoSQL();
46	GerarNomesColunas(res);
47	}catch(Exception e){
48	System.out.println("Erro Gerando Nome das Colunas");
49	}
50	}
51	}
52	
53	public boolean MatriculaExiste(String Mat){
54	try{
55	Comando.ExecutarComando("select * from alunos where Matricula = " + Mat);
56	ResultSet rs = Comando.getResultadoSQL();
57	return (rs.next());
58	}catch(Exception e){
59	System.out.println("Erro Pesquisando Matricula.");
60	return true;
61	}
62	
63	}
64	
65	
66	public void GerarNomesColunas(ResultSet rs){
67	try{
68	Rsmd = rs.getMetaData();
69	for (int i = 1; i <= Rsmd.getColumnCount(); i++)
70	NomesColunas[i-1] = Rsmd.getColumnName(i);
71	}catch(Exception e){
72	System.out.println("Erro Gerando Nome das Colunas");
73	}
74	}
75	
76	public String FormatoCampo(int Pos,String Valor){
77	String Saida=Valor;
78	try{
79	switch(Rsmd.getColumnType(Pos))
80	{
81	case Types.VARCHAR:Saida = "" + Valor +"";
82	break;
83	case Types.DATE: Saida = ""+Valor +"";
84	break;
85	case Types.INTEGER: Saida = Valor;
86	break;
87	case Types.CHAR: Saida = ""+Valor+"";

88	break;
89	case Types.FLOAT:Saida = Valor;
90	break;
91	}
92	return Saida;
93	}catch(Exception e) {
94	return Saida;
95	}
96	
97	}
98	
99	public void CadastrarDados(){
100	int i=0,itens=0,j=0;
101	String Escolha = "S";
102	String Mensagem = " Confirma o Cadastro dos Seguintes Dados: \n";
103	String Colunas="","Dados="","Operacao="";
104	String ValorLido="";
105	JOptionPane.showMessageDialog(null," Cadastro de ALUNOS \n"+
106	"Os campos com * são Obrigatórios!");
107	String Valores[] = new String[19];
108	
109	while (i < 19){
110	Valores[i] = JOptionPane.showInputDialog(null,Campos[i]);
111	ValorLido = Valores[i];
112	do{
113	if (i == 0 && MatriculaExiste(ValorLido)){
114	JOptionPane.showMessageDialog(null,"Este Número de Matrícula já foi usado!\n"+
115	"Informe um novo código para Matrícula");
116	ValorLido = JOptionPane.showInputDialog(null,Campos[i]);
117	Valores[i] = ValorLido;
118	}
119	}while(i==0 && MatriculaExiste(ValorLido));
120	
121	if(Valores[i].equals("") && (Campos[i].indexOf("*") != -1)){
122	JOptionPane.showMessageDialog(null," Este campo é obrigatório \n"+
123	"Por Favor, recadastre-o!");
124	}
125	else{
126	i++;
127	}
128	if (!ValorLido.equals(""))
129	itens++;
130	}
131	if (!Valores[18].equals("")){
132	if (Valores[18].charAt(0) == 'S' Valores[18].charAt(0) == 's')

SEGUIE ►

133	Valores[18] = "1"; //Sim
134	else
135	Valores[18] = "0"; //Não
136	}
137	j=0;
138	for(i=0;i<19;i++){
139	if (!Valores[i].equals("")){
140	Mensagem = Mensagem + Campos[i] + " : "+ Valores[i]+"\\n";
141	Colunas=Colunas+NomesColunas[i];
142	Dados =Dados +FormatoCampo(i+1,Valores[i]);
143	if (j < itens-1){
144	Colunas=Colunas + ",";
145	Dados = Dados + ",";
146	
147	}
148	j++;
149	}
150	}
151	Escolha = JOptionPane.showInputDialog(Mensagem + "\\n Sim(S) - Não(N)");
152	if (Escolha.equalsIgnoreCase("S")){
153	Operacao = "insert into alunos("+Colunas+") values (" +Dados+"");
154	try{
155	Comando.ExecutarComando(Operacao);
156	}catch(Exception e){
157	System.out.println(e.getMessage());
158	}
159	}
160	
161	}
162	
163	public void ExcluirPorMatricula(){
164	String Valores[] = new String[19];
165	String Escolha = "S";
166	String Saida = "";
167	String Mat = JOptionPane.showInputDialog("Informe a Matrícula para Exclusão:");
168	try{
169	Comando.ExecutarComando("select * from alunos where Matricula = " + Mat);
170	ResultSet rs = Comando.getResultadoSQL();
171	if (rs.next()){
172	Rsmc = rs.getMetaData();
173	for (int i = 1; i <= 2; i++) {
174	Valores[i-1] = rs.getString(Rsmc.getColumnName(i));
175	Saida = Saida + Campos[i-1]+" : "+Valores[i-1]+"\\n";
176	}

SEGUIE ►

177	}
178	
179	if (Saida.equals("")){
180	JOptionPane.showMessageDialog(null,"Matrícula Inexistente!");
181	}
182	else{
183	JOptionPane.showInputDialog("Confirma a exclusão do Aluno:\n"+
184	Saida+"\n"+
185	"Sim(S) - Não(N)");
186	if (Escolha.equalsIgnoreCase("S"))
187	Comando.ExecutarComando("Delete from alunos where Matricula = " + Mat);
188	}
189	}catch(Exception e){
190	System.out.println("Exclusão não pode ser feita!" + e.getMessage());
191	}
192	
193	}
194	
195	public void AlterarPorMatricula(){
196	int i;
197	String Valores[] = new String[19];
198	String Saida = "";
199	String Escolha = "S";
200	String ValorLido = "";
201	String Colunas = "";
202	String Dados = "";
203	String Operacao = "";
204	String Mat = JOptionPane.showInputDialog("Informe a Matrícula para Alteração:");
205	try{
206	Comando.ExecutarComando("select * from alunos where Matricula = " + Mat);
207	ResultSet rs = Comando.getResultadoSQL();
208	if (rs.next()){
209	Rsmd = rs.getMetaData();
210	for (i=1; i <= Rsmd.getColumnCount(); i++){
211	Valores[i-1] = rs.getString(Rsmd.getColumnName(i));
212	if (i == 19){
213	if(Valores[i-1].equals("1"))
214	Valores[i-1] = "Sim";
215	else
216	Valores[i-1] = "Não";
217	}
218	Saida = Saida + Campos[i-1]+":" +Valores[i-1]+"\n";
219	}

SEGUIE ►

220	}
221	
222	if (Saida.equals("")){
223	JOptionPane.showMessageDialog(null, "Matrícula Inexistente!");
224	}
225	else{
226	Escolha = JOptionPane.showInputDialog("Confirma a alteração do Aluno:\n"+
227	Saida+"\n"+
228	"Sim(S) - Não(N)");
229	if (Escolha.equalsIgnoreCase("S")){
230	i=2;
231	while (i<=19){
232	Valores[i-1] = JOptionPane.showInputDialog(null, Campos[i-1], Valores[i-1]);
233	if (Valores[i-1].equals("") && (Campos[i-1].indexOf("*") != -1)){
234	JOptionPane.showMessageDialog(null, " Este campo é obrigatório
235	"\n"+
236	" Por Favor, recadastre-o!");
237	}
238	else{
239	if (i==19 && (!Valores[i-1].equals(""))){
240	if (Valores[i-1].charAt(0) == 'S' Valores[i-1].charAt(0)
241	== 's')
242	Valores[i-1] = "1"; //Sim
243	else
244	Valores[i-1] = "0"; //Não
245	}
246	i++;
247	}
248	}
249	Operacao = "Update Alunos set ";
250	for (i=1; i<19; i++){
251	if (!Valores[i].equals("")){
252	Operacao = Operacao + NomesColunas[i] + "=" +
253	FormatoCampo(i+1, Valores[i]);
254	if (i < 18){
255	Operacao = Operacao + ",";
256	}
257	}
258	Operacao = Operacao + " where Matricula = " + Mat;
259	Comando.ExecutarComando(Operacao);
260	}
261	}catch(Exception e){
262	System.out.println("Alteração não pode ser feita!" + e.getMessage());
263	}
264	}
265	}



Síntese

Nesta unidade você pôde mais do que estudar, pôde praticar as técnicas de programação em Java para a criação de uma aplicação comercial com acesso a banco de dados.

O processo de implementação de uma solução computacional pode ser mais bem elaborado quando dividido em subprogramas, ou métodos de programação.

Dividindo-se o problema, fica simples de chegar a uma solução desejada. No caso da classe *Alunos*, os processos de cadastramento, exclusão e alteração foram solucionados de forma independente, para que problemas de desenvolvimento de um processo não influenciassem negativamente em outros.

Com isso, as soluções foram sendo projetadas e implementadas de forma segura e, principalmente, organizada.

Assim, a união das partes com certeza vai representar um todo.



Atividades de auto-avaliação

1. Qual a finalidade em se criar uma rotina de verificação para valores cadastrados em colunas do tipo chave primária?



Saiba mais

- HORSTMANN, C. S. **Big Java**. Porto Alegre: Artmed Bookman, 2004.
- THOMPSON, M. A. **Java 2 & Banco de Dados**. Erica, 2002.

Na Internet:

- <http://www.guj.com.br>
- <http://www.mundojava.com.br>

Criando as consultas para a Tabela de Alunos



Objetivos de aprendizagem

- Aplicar a estrutura do comando *Select* nas pesquisas de banco de dados.
- Implementar e testar os processos de pesquisa de dados da Tabela de Alunos.



Seções de estudo

- Seção 1** Estrutura atual da classe de cadastro de alunos.
- Seção 2** Comando de seleção.
- Seção 3** Preparando a rotina de consulta.
- Seção 4** Menu de acesso.



Para início de conversa

Nesta unidade você dará continuidade ao processo de implementação das rotinas de manipulação da Tabela de Alunos do banco de dados “Sabe Tudo”.

Talvez você se pergunte por que essas rotinas não foram apresentadas na unidade anterior. Pois bem, a finalidade de separar os processos de cadastramento, exclusão e alteração de dados do processo de seleção é apenas para facilitar a sua compreensão, nada mais que isso.

Como você conhece o processo de inserção e manipulação de dados, agora é hora de montar as rotinas de seleção de dados a fim de fechar todas as regras de negócio referentes à Tabela de Alunos.

Preparado?

Acredito que sim, então, vamos lá.

Seção 1 – Estrutura atual da classe de cadastro de alunos

Antes de partir para as rotinas de seleção de dados, é essencial que você tenha em mente a estrutura da classe **Alunos**, que foi apresentada e implementada na unidade anterior. Na tabela a seguir, é apresentada a estrutura da classe Alunos com seus atributos e métodos de trabalho.

Alunos
Atributos
<i>ResultSetMetaData Rsmd;</i>
<i>ComandosSQL Comando;</i>
<i>String NomesColunas[]</i>
<i>String Campos[]</i>
Métodos
<i>boolean MatriculaExiste(String Mat)</i>
<i>void GerarNomesColunas(ResultSet rs)</i>
<i>String FormatoCampo(int Pos,String Valor)</i>
<i>void CadastrarDados()</i>
<i>void ExcluirPorMatricula()</i>
<i>void AlterarPorMatricula()</i>

Apesar de se tratar de uma estrutura simples, essa classe tem rotinas bem funcionais, como você deve ter notado ao implementá-la. Caso você não tenha implementado a classe, seria muito bom que você realizasse essa tarefa para interagir diretamente com os códigos em Java, e mais do que isso, para que você insira na implementação o seu toque pessoal, com as rotinas que você está acostumado a fazer.

As implementações apresentadas aqui e na unidade anterior possuem as características da minha forma de programar e com certeza em muitos casos você se indagou: “Eu faço diferente!”.

Isso é mais do que normal, por isso é muito importante que você implemente o código e verifique o que você pode fazer de diferente, personalizando a sua implementação. Só não esqueça que o resultado final tem que ser o mesmo, ou seja, funcionar de acordo com as regras de negócio do sistema.

Seção 2 – Comando de seleção

De forma bem resumida, é necessário que você compreenda o funcionamento de um comando de seleção. Basicamente, um comando de **Select** representa duas informações:

- as colunas que se deseja selecionar;
- as linhas que atendem o filtro da seleção.

Por isso, a sintaxe mais simples de um comando **Select** é:

```
Select Colunas from Tabela  
where Condição Lógica;
```

Um comando **Select** é um comando de retorno, sendo assim, sempre haverá uma informação de saída, nem que seja para indicar que nenhum registro atendeu as condições de filtro do comando, ou seja, o retorno foi “**nenhuma linha selecionada**”.



O que pode variar num comando **Select**?

Muito pouco. Partindo do princípio que se está selecionando dados de uma mesma tabela, o que pode variar é:

- o nome das colunas selecionadas;
- os nomes das colunas usadas na geração da condição de filtro.

Por exemplo, usando a Tabela de Alunos como referência, pode-se montar um comando de seleção da seguinte forma:

```
Select ? from Alunos where ? = ?.
```

– *Calma, não se assuste!*

Apenas pense que no lugar de cada símbolo de “?” pode ser inserida uma informação de acordo com o comando de seleção desejado.

Selecionar todos os dados dos alunos chamados “Marcelo”:

```
Select _ from Alunos where Nome = “Marcelo”.
```

Selecionar todos os dados dos alunos que residem em “Palhoça”:

```
Select _ from Alunos where Cidade = “Palhoça”.
```

Sendo assim, é possível chegar à conclusão que o comando **Select** precisa de três parâmetros para funcionar:

- nome das colunas;
- nome do campo de filtro;
- valor do campo de filtro.

E é essa a regra que será adotada para a criação da rotina de consulta na base de dados de alunos, que será implementada em Java por meio de um método com a seguinte estrutura:

```
public void Consultar(String Colunas, String Campos, String Valores).
```

Veja sobre esse assunto na próxima seção.

Seção 3 – Preparando a rotina de consulta

Nesta seção você terá contato com as rotinas necessárias para a geração da sintaxe do comando **Select** desejado. É preciso, primeiro, apresentar duas estruturas utilizadas na classe Alunos:

Nomes Colunas	Matricula	Nome	...	Grupo_Sangue	Doador
---------------	-----------	------	-----	--------------	--------

Essa estrutura representa o nome físico das colunas da Tabela de Alunos, é utilizada para definição dos campos que serão selecionados por meio do comando **Select** e não possui interação com usuário do sistema, pois ele não precisa saber os nomes das colunas para conseguir utilizar o sistema.

A outra estrutura é a seguinte:

Campos	Matrícula *	Nome *	...	Grupo_Sanguíneo	Doador de Órgãos
--------	-------------	--------	-----	-----------------	------------------

Essa estrutura representa os nomes de cada campo da Tabela de Alunos, porém na forma que será apresentada ao usuário, inclusive com a grafia correta.

Essas duas estruturas trabalham em paralelo, pois utilizam os mesmos índices de armazenamento, veja:

	0	1		17	18
Nomes Colunas	Matricula	Nome	...	Grupo_Sangue	Doador
Campos	Matricula *	Nome *	...	Grupo_Sanguíneo	Doador de Órgãos

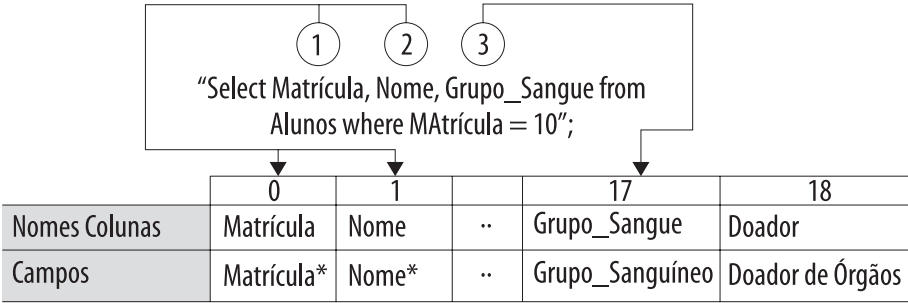
Dessa forma, um comando de seleção com a sintaxe:

“Select Matricula, Nome, Grupo_Sangue from Alunos where Matricula = 10”;

Esse comando de seleção utilizará os nomes de colunas e de campos das posições 0,1 e 17. Entretanto, para o comando de seleção, executado pelo banco de dados e que retornará ao Java os dados que atendem a condição de pesquisa, as colunas Matricula, Nome e Grupo_Sangue representam as posições 1, 2, 3.

Para ajudar na sua compreensão, veja o diagrama a seguir:

GRÁFICO 9.1 - FORMA DE REPRESENTAÇÃO DE UM SELECT NA CLASSE ALUNOS



Para tornar essa forma de representação simples, independente do comando *Select* gerado, é preciso que ao se especificar um comando de seleção, a aplicação em Java identifique os nomes das colunas físicas e de apresentação, com seus respectivos índices e o valor representativo no retorno do *Select*.

Para isso, serão necessários dois novos métodos na classe de Alunos. O primeiro método, chamado de **Posição**, retorna a posição do nome físico da coluna selecionada em relação à estrutura de Nomes de Colunas.

O código em Java ficará assim:

1	public int Posicao(String Item){
2	int i=0;
3	while((i < 19) && (!NomesColunas[i].equalsIgnoreCase(Item)))
4	i++;
5	return (i+1);
6	}

Esse método recebe um item como parâmetro. Por meio de um laço condicional se faz a comparação de todas as palavras armazenadas no vetor NomesColunas com o valor do item informado. O laço se encerra quando todas as posições já tiverem sido comparadas ou quando o item procurado é igual a palavra armazenada no vetor.

Para cada item pesquisado, o sistema precisa saber qual a sua posição dentro do vetor NomesColunas. Sendo assim, para cada coluna selecionada é preciso armazenar o seu índice na estrutura NomesColunas.

O objetivo principal é saber quais colunas fazem parte do *Select* montado, entre as 19 colunas que compõem a Tabela de Alunos.

No caso do *Select* :

“Select Matricula, Nome, Grupo_Sangue from Alunos where Matricula = 10”;

O resultado seria:

	0	1		17	18
Nomes Colunas	Matricula	Nome	...	Grupo_Sangue	Doador

Ou seja, das 19 colunas da tabela de Alunos, os índices 0,1 e 17 estão sendo requisitados. Sendo que a coluna Matrícula é representada pela expressão “Matrícula * “, a coluna Nome pela expressão “Nome * “ e a coluna Grupo_Sangue pela expressão “Grupo Sanguíneo”.

Para armazenar os índices das colunas requisitadas será implementado o seguinte método:

1	public Vector MontarIndices(String Colunas){
2	Vector Indices = new Vector();
3	String Coluna="";
4	Colunas = Colunas + ',';
5	for (int i=0;i< Colunas.length();i++){
6	if (Colunas.charAt(i) != ','){
7	Coluna = Coluna + Colunas.charAt(i);
8	}
9	else{
10	Indices.addElement(Posicao(Coluna)+"");
11	Coluna = "";
12	}
13	}
14	return Indices;
15	}

Esse método recebe o nome das colunas que serão selecionadas, algo como “Nome, Matricula,Cidade” e retorna um vetor dinâmico do tipo Vector.

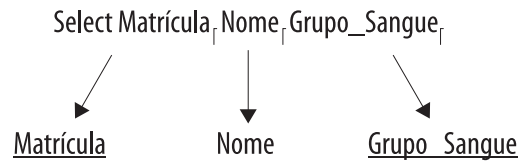


Esse **vetor** é chamado de dinâmico porque o seu tamanho é definido durante a execução do programa a cada nova inserção de um elemento.

Para montar os índices, é realizado um laço que varre caractere por caractere do parâmetro Colunas, que representa cada coluna selecionada, porém separadas por vírgula - com exceção da última coluna que não possui uma vírgula no final. Por isso o método insere no final do parâmetro Colunas uma vírgula (Colunas = Colunas + ',');

Veja a representação a seguir:

GRÁFICO 9.2 - SEPARADOR DE COLUNAS SELECIONADAS



Toda vez que uma vírgula é encontrada como símbolo dentro do parâmetro `Colunas`, significa que uma coluna selecionada foi definida. Pelo método posição é inserido no vetor de índices a posição em que essa coluna se encontra dentro do vetor de **NomesColunas**.

Como a definição dos campos que serão buscados é uma interação direta com o usuário do sistema, é preciso uma interface na qual o usuário especifique quais os campos serão selecionados, porém de forma amigável.

A seguir é apresentado um código-fonte que implementa a interface de solicitação dos campos que definirão o comando *Select*.

1	public String SelecionarItens(){
2	char Continua = 'S', Escolha = 'S';
3	int i=0;
4	String Escolhidas = "";
5	do{
6	if ((Escolhidas.length() > 0) && (Escolha == 'S' Escolha == 's'))
7	Escolhidas = Escolhidas + ',';
8	Escolha = JOptionPane.showInputDialog(
9	"Selecionar o campo: " +
10	Campos[i] +
11	"\n S-Sim N-Não","S").charAt(0);
12	if(Escolha == 'S' Escolha == 's'){
13	Escolhidas = Escolhidas + NomesColunas[i];
14	}
15	i++;
16	Continua = JOptionPane.showInputDialog(
17	"Deseja Sair? Sim(S) - Não (N)","N").charAt(0);
18	}while (i < 19 && (Continua == 'N' Continua == 'n'));
19	return Escolhidas;
20	}

Esse método solicita uma confirmação do usuário com relação aos campos da tabela que ele deseja visualizar no resultado da execução do *Select*. Para cada campo é apresentada uma tela, com o nome do campo e a solicitação se o mesmo deve ser incluído na seleção. Por padrão, a resposta vem sempre com o valor Sim (S).

Como os campos são separados por vírgula, a cada inserção de uma coluna de pesquisa, uma vírgula é incluída ao lado. O processo se repete até que todos os campos sejam informados ou o usuário cancele a operação, que por padrão vem com o valor Não (N).

Com essas rotinas prontas, o próximo passo é a criação do método de seleção, que possui o seguinte código:

1	public void Consultar(String Colunas, String Campo,String Valor){
2	String Saida = "";
3	Vector Seleccionados = MontarIndices(Colunas);
4	char Continua = 'S';
5	String Valores[] = new String[19];
6	int i=0,pos=0;
7	try{
8	Comando.ExecutarComando("select " + Colunas +
9	" from alunos where " + Campo +
10	" like '%" + Valor + "%");
11	ResultSet rs = Comando.getResultadoSQL();
12	while ((Continua == 'S' Continua == 's') && (rs.next())){
13	Saida = "";
14	Rsmd = rs.getMetaData();
15	i=1;
16	pos=0;
17	while(i < 19){
18	if (Seleccionados.contains(i+"")){
19	Valores[pos] = rs.getString(Rsmd.getColumnName(pos+1));
20	if (i == 19 && Seleccionados.contains(i+"")){
21	if(Valores[pos].equals("1"))
22	Valores[pos] = "Sim";
23	else
24	Valores[pos] = "Não";
25	}
26	Saida = Saida + Campos[i-1]+":"+Valores[pos]+"\\n";
27	pos++;
28	}
29	i++;
30	}
31	JOptionPane.showMessageDialog(null,Saida);
32	Continua = JOptionPane.showInputDialog(
33	"Continuar a Listagem? Sim(S) - Não (N)").charAt(0);
34	}
35	if (Saida.equals("")){
36	JOptionPane.showMessageDialog(null,Campo + " Inexistente!");
37	}
38	
39	}catch(Exception e){
40	System.out.println("Consulta não pode ser feita!" + e.getMessage());
41	}
42	}

Esse método permite a consulta de qualquer registro na Tabela de Alunos. Para que ele funcione são necessários os parâmetros **colunas**, **campo de filtro** e **valor de filtro**.

O método apresenta na telas os campos solicitados pelo usuário da aplicação e o valor de cada campo. O processo se repete até que todos os cadastros sejam visualizados ou o cliente cancele a operação.

Esse método é destinado às pesquisas que envolvem discriminação de quais colunas serão apresentadas na tela, caso se deseje a pesquisa de todos os campos da tabela, o método para isso possui o seguinte código:

1	public void ConsultarTodos(){
2	String Saida = "";
3	char Continua = 'S';
4	String Valores[] = new String[19];
5	int i=0;
6	try{
7	Comando.ExecutarComando("select * from alunos");
8	ResultSet rs = Comando.getResultadoSQL();
9	while ((Continua == 'S' Continua == 's') && (rs.next())){
10	Saida = "";
11	Rsmd = rs.getMetaData();
12	for (i=0; i < Rsmd.getColumnCount();i++){
13	Valores[i] = rs.getString(Rsmd.getColumnName(i+1));
14	if (i == 18){
15	if(Valores[i].equals("1"))
16	Valores[i] = "Sim";
17	else
18	Valores[i] = "Não";
19	}
20	Saida = Saida + Campos[i]+"."+Valores[i)+"\n";
21	
22	}
23	JOptionPane.showMessageDialog(null,Saida);
24	Continua = JOptionPane.showInputDialog(
25	"Continuar a Listagem? Sim(S) - Não (N)","S").charAt(0);
26	}
27	}catch(Exception e){
28	System.out.println("Consulta não pode ser feita!" + e.getMessage());
29	}
30	}

Seção 4 – Menu de acesso

Para facilitar a interação entre o usuário do sistema e as regras de negócio da manipulação de dados da Tabela de Alunos, é essencial um *menu* de opções. A criação desse *menu* é feita pelo seguinte código:

1	public void Menu(){
2	int op=0;
3	String ColunasPesq = "";
4	String Procura = "";
5	do{
6	op = Integer.parseInt(JOptionPane.showInputDialog(" Cadastro de Alunos \n"+
7	"1-Cadastrar\n"+
8	"2-Excluir\n"+
9	"3-Alterar\n"+
10	"4-Consultar por Nome\n"+
11	"5-Consultar por Matrícula\n"+
12	"6-Consultar Todos\n"+
13	"7-Sair","7")));
14	switch (op){
15	case 1:CadastrarDados();
16	break;
17	case 2:ExcluirPorMatricula();
18	break;
19	case 3:AlterarPorMatricula();
20	break;
21	case 4:ColunasPesq = SelecionarItens();
22	Procura = JOptionPane.showInputDialog("Nome para Pesquisa:");
23	Consultar(ColunasPesq,"Nome",Procura);
24	break;
25	case 5:ColunasPesq = SelecionarItens();
26	Procura = JOptionPane.showInputDialog("Matrícula para Pesquisa:");
27	Consultar(ColunasPesq,"Matricula",Procura);
28	break;
29	case 6:ConsultarTodos();
30	break;
31	}
32	}while(op != 7);
33	}

Esse método implementa uma chamada para cada um dos métodos vistos anteriormente, permitindo a interação do usuário com todas as funcionalidades do sistema. Essa chamada terá a seguinte interface:

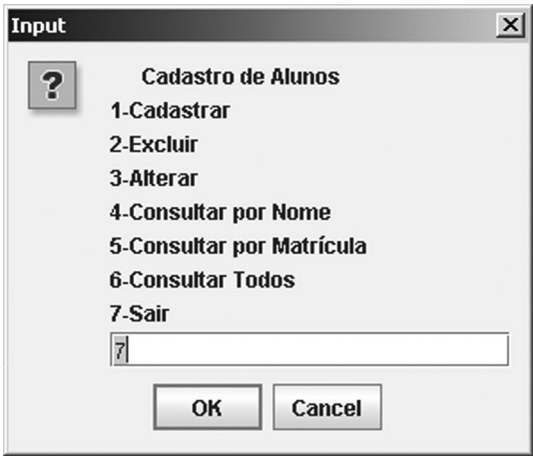


FIGURA 9.1 - TELA DE MENU

Para usar a classe *Alunos* é preciso que você crie uma nova classe que instancie a classe *Alunos* e execute a chamada do método *Menu*.

Veja:

1	<code>class Dados_Alunos{</code>
2	<code>public static void main(String args[]){</code>
3	<code>Alunos A = new Alunos();</code>
4	<code>A.Menu();</code>
5	<code>System.exit(0);</code>
6	<code>}</code>
7	<code>}</code>



Síntese

Nesta unidade você pôde praticar as técnicas de programação em Java para a criação de métodos que permitem a seleção de dados em tabelas.

A base de funcionamento dos métodos está fundamentada na sintaxe do comando *Select*, que pode ser dividido em: **colunas selecionadas** e **condição para seleção**.

O principal objetivo desses métodos implementados é permitir que os comandos de seleção sejam criados sem que o usuário do sistema tenha que saber como um comando em SQL funciona, isso deve ser transparente para ele, porém deve ser dominado pelo programador do *software*.



Atividade de auto-avaliação

1. O processo de implementação de uma consulta em Java pode ser considerado um gerador de comandos?



Saiba mais

DEITEL, H. M.; DEITEL, P. J. **Java**: como programar. 4. ed. Bookman, 2003.

HORSTMANN, C. S. **Big Java**. Artmed Bookman, 2004.

THOMPSON, M. A. **Java 2 & Banco de Dados**. Erica, 2002.



Para concluir o estudo

Caro aluno,

Nesta disciplina a minha intenção foi apresentá-lo ao mundo dos sistemas de bancos de dados, possibilitando a você a compreensão do seu papel como um profissional que está envolvido com armazenamento e gerenciamento de informações.

Porém, além de gerenciar dados é necessário que você compreenda como funciona a integração de uma ferramenta de banco de dados e um sistema desenvolvido em linguagem de programação.

Acredito que para alguns os comandos e tecnologias aqui apresentadas talvez não fossem novidades, no entanto espero que a disciplina tenha contribuído para um aprimoramento ao seu conhecimento, independente dos seus conhecimentos adquiridos anteriormente.

Tenho certeza que esta disciplina também permitiu um amadurecimento no seu conhecimento da linguagem de programação em Java e principalmente nos conceitos da programação orientada a objetos.

Acredito que muitas perguntas ainda se formam na sua mente, porém, este é o meu papel como autor, fomentar a curiosidade e estimular a busca por novos conhecimentos.

Espero que através das ferramentas disponibilizadas pelo ambiente de ensino a distância você possa “baixar” o código fonte de toda a solução proposta aqui e tenha a capacidade de dar um toque pessoal na solução.

– Quem sabe, você começa uma nova carreira de programador e analista de sistemas?

Espero ter atingido pelo menos o objetivo de plantar a semente da busca pela informação.

Parabéns, você acaba de dar mais um grande passo rumo ao objetivo maior que é se tornar um profissional que domine a tecnologia de banco de dados e tenha a capacidade de desenvolver soluções computacionais que atendam as necessidades de um mercado tão competitivo.

Se a caminhada parecia longa, que bom, você acaba de encurtá-la!

Desejo que esta sua caminhada te aproxime cada vez mais do seu objetivo maior, mesmo que em alguns momentos o final da caminhada se apresente mais distante, levante a cabeça, tome mais fôlego, e rumo ao sucesso.

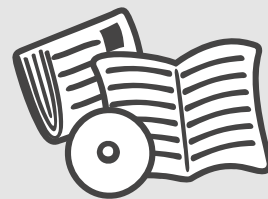
Professor Marcelo Medeiros

Sobre o professor conteudista



Marcelo Medeiros é bacharel em Ciências da Computação pela Universidade Regional de Blumenau e professor da Unisul desde 1995, na qual leciona as disciplinas de Banco de Dados, Programação e Estrutura de Dados para os cursos de Engenharia, Ciência da Computação e Sistemas de Informação. Atualmente participa do projeto de Incubadora da Unisul, desenvolvendo projetos de software na área de automação e segurança via IP. Atua como consultor junto a empresas no desenvolvimento de novos produtos de informática.

Referências



ANSELMO, Fernando. **Tudo que você queria saber sobre JDBC**. Visual Books: 2001.

_____. **Aplicando Orientação a Objetos em Java**. Visual Books: 2005.

COSTA, Luis Carlos da. **Java para Iniciantes**. Ciência Moderna: 2002.

CHEN, Peter. **Modelagem de dados: a abordagem entidade-relacionamento para projeto lógico**. São Paulo: Makron Books, 1990.

DATE, C J. **Bancos de dados: fundamentos**. Rio de Janeiro: Campus, 1985.

_____. **Introdução a sistema de banco de dados**. 8. ed. Rio de Janeiro: Campus, 1990.

DEITEL, H.M.; DEITEL, P.J. **Java-Como Programar**. Sexta edição. Ed. Pearson, 2005.

FILHO, Trajano Leme. **Metodologia de Desenvolvimento de Sistema**. São Paulo: Axcel, 2003.

HORSTMANN, Cay S. **Big Java**. Porto Alegre: Bookman, 2004.
MECENAS, Ivan. **Fundamentos Swing e JDBC: 2ª edição**. Alta Books: 2005.

NIEDERAUER, Juliano. PRATES, Rubens. **Mysql – Guia de Referência Rápida**. São Paulo: NovaTec, 2005.

RANGEL, Alexandre. **Mysql-Projeto Modelagem e Desenvolvimento de Banco de Dados**. Alta Books: 2005.

SANTOS, Rui Rossi dos. **Programando em Java 2-Teorias e Aplicações**. Axcel: 2004.
SIERRA, Kathy; BATES, Bert. **Java Use a cabeça**. Alta Books: 2005.

SUHRING, Steve. **Mysql – A Bíblia**. Rio de Janeiro: Campus, 2002.

THOMPSON, Marco Aurélio. **Java 2 & Banco de Dados**. São Paulo, Érica: 2002.

Respostas e comentários das atividades de auto-avaliação



Unidade 1

1. A maioria das profissões exige que o especialista seja formado em um curso de graduação, como médicos e advogados. Na área de desenvolvimento de softwares não exigência de graduação para que um profissional execute as suas atividades. Qual a sua opinião sobre o tema?

Resposta:

Este tema vem sendo discutido há muito tempo por especialistas da área e tem se tornado um tema polêmico. Há grupos a favor e contra, sem que se chegue a um acordo. Há explanações de ambos os grupos para justificar a sua escolha.

Os grupos que são a favor da liberação da atividade de desenvolvimento de software sem a exigência de graduação, ou diploma, se baseiam na capacidade de profissionais de outras áreas em desenvolver ferramentas na área da computação.

Outra justificativa muito forte é o fato deles deterem o conhecimento sobre os assuntos que não são da área de informática, mas que precisam ser informatizados.

Já os grupos que exigem o diploma, se baseiam no investimento que um aluno faz durante a sua graduação e a importância da profissão como justificativa para se exigir o diploma.

As profissões relacionadas à informática são novas, e estão em constante evolução. Talvez por se tratar de uma atividade nova, que integra muitas outras atividades, que ainda não se tenha chegado a uma conclusão sobre a obrigatoriedade do diploma ou não.

Independente de graduação, diploma, o profissional que desenvolve softwares deve ter em mente que o seu trabalho é tão importante quanto outro qualquer, e necessita de muita dedicação, estudos e uma constante atualização, desde equipamentos a ferramentas computacionais, pois é um mercado seletivo.

2. Quando se desenvolve um produto, como uma cadeira, uma casa, etc., durante o processo de criação é possível apresentá-lo ao cliente, para seu acompanhamento. Porém, no caso de desenvolvimento de softwares não há um produto palpável para ser apresentado. Como agir em uma situação como esta?

Resposta:

O desenvolvimento de software possui muitas particularidades, principalmente porque não existe um produto que possa ser apresentado ao cliente. O desenvolvimento possui etapas bem definidas, e até que se chegue a implementação, não há o que se apresentar ao cliente.

Desta forma, a relação entre desenvolvedores e cliente deve ser de transparência, confiança e cordialidade. Ele deve se sentir seguro com relação ao que ele contratou e com o que ele terá como produto final.

Durante a fase de implementação é muito comum que se apresente ao cliente pequenas versões do produto final para sua prévia aprovação. Estas pequenas versões são chamadas de protótipo, que além de verificar se o projeto está no rumo certo, permitem que o cliente tenha um contato mais direto com o produto esperado.

Unidade 2

1. A utilização de script para criação de banco de dados não é obrigatório, porém é muito utilizado pelos profissionais de banco de dados. Sendo assim, o script deve possuir algumas vantagens, você poderia citar algumas?

Resposta:

- Organização das tabelas;
- Criação automática das tabelas do banco de dados;
- Facilidade em executar várias vezes os comandos em SQL;
- Facilidade em realizar alterações no modelo de banco de dados;
- O modelo de banco de dados se mantém atualizado e refletindo a realidade.

2. A alteração do modelo de dados deve gerar, conseqüentemente, uma alteração nos script de criação do mesmo, pois, caso contrário, o modelo não representará fielmente o banco de dados utilizado. Esta afirmação esta correta? Justifique a sua resposta.

Resposta:

O banco de dados físico, criado a partir de um script, ou não, deve representar fielmente as regras e tabelas especificadas no modelo de

dados. O modelo de dados representa graficamente a forma na qual os dados são armazenados e estruturados no modelo físico. Sendo assim, o modelo de dados é um espelho do banco de dados criado. Se a criação de um banco de dados é realizada a partir de um script, qualquer alteração no modelo deve ser atualizada no script, e vice-versa.

3. O MySQL permite a execução de um script de criação de tabelas de um banco de dados sem que tenha sido especificado antes qual o database usado? Caso isso não seja possível, como garantir que as tabelas sempre serão criadas no database correto ao se executar o script?

Resposta:

Como o database representa um conjunto de tabelas, caso o usuário tente criar um conjunto de tabelas sem especificar em qual database, o MySQL não permite a criação do modelo de dados. Para evitar que este problema ocorra, a melhor solução é inserir no script de criação do modelo, na primeira linha o comando use seguido do database que deve ser aberto. Desta forma, a criação do modelo ocorrerá sempre no local correto.

Exemplo: use SabeTudo;

Unidade 3

1. O Java é uma linguagem de programação atual, moderna e muito utilizada. O profissional que é um especialista nesta linguagem de programação possui grandes chances de atuar no mercado atual. Sendo assim, o especialista nesta linguagem não precisa se preocupar, pois estará sempre garantido no mercado, dado que é muito difícil uma linguagem de programação deixar de ser usada, sobretudo a linguagem Java. Esta afirmação é correta? Qual a sua opinião?

Resposta:

Não. Qualquer profissional, independentemente da área de atuação, precisa se atualizar. Na área de informática isto se torna ainda mais verdade, pois é uma área nova, em que as mudanças acontecem de forma constante e rápida. A cada instante novas tecnologias e metodologias surgem, e o profissional que não estiver atualizado, passa a ficar obsoleto. São muitas as tecnologias que surgiram como a melhor solução, e pouco tempo depois, sumiram do mercado, levando com elas os profissionais que tinham ali a única forma de exercer as suas atividades.

Unidade 4

1. Uma característica fundamental num driver de conexão com banco de dados é a sua transparência, ou seja, quanto maior for a sua abstração para o programador, melhor a ferramenta. Sendo assim, pode-se afirmar que o programador e o especialista em banco de dados não precisam saber como funciona o driver, mas sim, saber como tirar o melhor proveito dele. Esta afirmativa é correta? Justifique sua resposta.

Resposta:

Sim, correta. Quanto maior o nível de abstração do driver de conexão, melhor para o programador e para o especialista de banco de dados. Estes dois profissionais não precisam saber como o driver foi implementado, qual linguagem foi usada, apenas precisam ter certeza de que o driver é confiável e funciona de acordo com as suas necessidades.

2. O uso das Fontes de Dados é uma forma de abstração para o programador?

Resposta:

Sim. A partir do uso da fonte de dados, o programador não precisa saber o local em que o banco de dados está conectado, quem é o usuário administrador e a senha de acesso. Tudo isso é abstraído pelo uso de um apelido para fonte de dados, o que tende a facilitar a sua vida durante o processo de implementação.

Unidade 5

1. O uso de ferramentas que auxiliam o profissional de banco de dados com interação com a análise de sistemas e programação, as chamadas ferramentas CASE, podem facilitar o processo de implementação a ponto que não necessite de um especialista?

Resposta:

Acredito que não, a maioria das chamadas ferramentas CASE é um auxílio ao especialista. Todo o processo de análise de requisitos e funcionalidades de um sistema ainda dependem muito do conhecimento e da experiência do especialista. Chego a dizer que um especialista realiza as suas tarefas sem a ferramenta CASE, já a ferramenta CASE não faz nada sem o especialista.

2. Seria correto afirmar que o processo de compilação de um arquivo em Java está diretamente relacionada ao código-fonte e a execução diretamente relacionada ao arquivo resultado da compilação, chamado de .class?

Resposta:

Com certeza. Um programa em Java é formado pelo código fonte, que obedece as regras de sintaxe e semântica da linguagem, representados pela extensão .java. Quando da compilação de um programa em Java, através do comando Javac, que na verdade é a execução do arquivo javac.exe, existente na pasta Bin do diretório em que está instalado o Java, o resultado se tudo estiver correto é um arquivo com mesmo nome, porém com a extensão .class.

Este arquivo .class é que pode ser executado pela máquina virtual do Java, através do arquivo java.exe também localizado na pasta Bin da instalação do Java, representado de forma resumida como o comando java do prompt do Windows.

Unidade 6

1. Qual a vantagem em se apresentar na tela, através do método getMessage(), o erro de uma Exception, como no exemplo a seguir?

```
try{
    //faz algo;
}catch(Exception e){
    System.out.println("O erro foi : " + e.getMessage());
}
```

Resposta:

O tratamento de erro através do try-catch no Java é uma ferramenta essencial e em muitos casos obrigatória. Diferente de algumas linguagens em que o tratamento de exceção é uma opção do programador, no Java isto não existe. Sendo assim, cabe ao programador utilizar este recurso da melhor forma possível. Através do método getMessage() é possível apresentar ao usuário do sistema uma mensagem mais significativa, que possa indicar o problema, do que simplesmente uma mensagem do tipo, houve erro. Mesmo que a mensagem do método getMessage() seja em inglês.

2. Na definição de uma classe em Java, é possível atualizar o valor de um atributo diretamente, sem que se use o seu método modificador. Esta forma de implementação pode ser considerada equivocada?

Resposta:

Veja o exemplo a seguir:



O uso dos métodos modificadores não é uma mera formalidade do Java. Apesar da linguagem aceitar que um atributo da classe receba um valor diretamente, isto deve ser evitado. O principal papel de um modificador é alterar o valor de um atributo, e deve ser esta a única forma de modificação utilizada. Além disto, o modificador tem a função de garantir a integridade dos dados, como por exemplo, evitar que idades negativas sejam cadastradas. Se o valor for alterado sem o uso do modificador, a regra de integridade estará desfeita.

Veja o exemplo:

```

public void setIdade(int _ldade){
    if(_ldade >=0)
        Idade = _ldade;
    else
        Idade = 0;
}

```

Unidade 7

1. Cite exemplos de metadados em banco de dados relacional.

Resposta:

São exemplos de metadados em um banco de dados relacional: nome do banco de dados, nomes das tabelas e seus atributos, relacionamentos (chaves primárias e estrangeiras) e tamanho das tabelas (número de linhas e colunas).

2. De que forma a linguagem Java obtém informações sobre os metadados de um SGDB?

Resposta:

Por meio do Driver JDBC. Esse pacote permite que o programador se conecte ao banco de dados e possa manipulá-lo. Na verdade, trata-se de um alto nível de abstração, pois ao usar classes como `DatabaseMetaData` o usuário está interagindo diretamente com os metadados do SGDB usado, porém ele nem fica sabendo disto.

Unidade 8

1. Qual a finalidade em se criar uma rotina de verificação para valores cadastrados em colunas do tipo chave primária?

Resposta:

O cadastramento de valores repetidos em colunas do tipo chave primária geram um erro grave que pode encerrar o programa de forma inesperada. Problemas como esse podem gerar certa desconfiança do usuário com relação à funcionalidade do sistema e das suas garantias de funcionamento. Além disso, o sistema de banco de dados não permitirá o cadastramento de dados repetidos em colunas com restrição de chave primária, sendo assim, não há necessidade de forçar esse tipo de operação para que o Java gere um tratamento de exceção.

Unidade 9

1. O processo de implementação de uma consulta em Java pode ser considerado um Gerador de Comandos?

Resposta:

Sim. Pois para fazer com que uma rotina em Java crie uma seleção a partir dos dados informados pelo usuário é necessário que o programador tenha um bom conhecimento sobre a sintaxe do comando SQL utilizado. A partir dos dados informados pelo usuário o sistema terá que criar o comando em SQL que represente fielmente a seleção desejada, e de forma transparente, pois o usuário não precisa saber como um comando em SQL funciona. Sendo assim, os dados informados pelo usuário precisam se transformar em um comando em SQL, de forma automática, eficiente e correta, caso contrário o sistema não funcionará.

