

Minicurso de Banco de Dados

Mário Leite

Agradecimentos

Agradeço primeiramente a Deus que me deu força e persistência para produzir mais este material, com muita pesquisa e dedicação.

Agradeço aos professores *Cláudio Torres* e *Evelline Soares Correia* da UNIUBE, pelo apoio e incentivo na elaboração deste material.

Um agradecimento especial ao prof. Edinei Campos da “R4 Informática” pelas preciosas dicas sobre segurança em servidores.

Organização do Material

Capítulo 1 - Sobre Tecnologia da Informação

Neste capítulo são apresentados, de maneira introdutória, os assuntos relacionados à manipulação de dados para se obter informações gerenciais nas tomadas de decisão. São apresentados os recursos básicos que compõem as Tecnologias de Informação (TI) e a importância de suas corretas manipulações.

Capítulo 2 - Tratamentos de Dados

Este capítulo focaliza de maneira mais profunda a natureza das manipulações dos dados armazenados em bancos de dados. É apresentado o conceito de Sistema Gerenciador de Bancos de Dados (SGBD) com suas características básicas e sua importância na manipulação e extração dos dados armazenados. São apresentados os tipos de bancos de dados, suas características e funcionalidades práticas no controle dos dados. O enfoque maior é dado sobre os bancos relacionais, suas características e praticidades de utilização nas empresas. Como consequência, também são estudados os elementos de um banco relacional: tabelas, campos, chaves (PK, FK) e índices. Tipos de entidades (forte, fraca, associativa) em função do relacionamento e cardinalidade são apresentadas de modo prático e bem acessível. O modelo de dados focalizado é o Modelo Entidade Relacionamento (MER) para estudar os relacionamentos entre as entidades.

Capítulo 3 - Normalização

Neste capítulo é apresentado o conceito e as práticas sobre Normalização: o objetivo e a importância de se criar uma base de dados normalizada e bem comportada. São apresentadas cinco formas normais: 1FN, 2FN, 3FN, 4FN e 5FN. Todas elas com exemplos práticos e esquemas bem explicados.

Capítulo 4 - Linguagem de Consulta Estruturada (SQL)

Neste capítulo é apresentada a técnica de consulta em bancos de dados baseada em instruções SQL (*Structured Query Language*), subdivididas em linguagem DDL (*Data Definition Language*) e DML (*Data Manipulation Language*). São estudados os diversos elementos de uma instrução sql: comandos, cláusulas, predicados, operadores e funções agregadas. São mostrados exemplos práticos de uso de instruções sql na execução de consultas.

Capítulo 5 - Servidores de Bancos de Dados

Este último capítulo apresenta de maneira bem prática o servidor Firebird; como baixar, instalar e configurar. São mostrados vários exemplos com bancos de dados criados no servidor através de instruções sql puras e também com o auxílio de um gerenciador gráfico: o IBExpert. Os elementos mais importantes de um banco de dados são estudados: tabelas, chaves, índices, *triggers*, *exceptions*, *stored procedures* e *generators*. Todos esses elementos são estudados passo-a-passo com figuras e esquemas bem esclarecedores. Também neste capítulo é mostrado como tornar o servidor mais seguro, criando novos usuários e alterando a senha-mestre.

Sumário

Capítulo 1 - Sobre Tecnologias da Informação

1.1 - Introdução	7
1.2 - Recursos e Tecnologias dos Sistemas de Informação	10

Capítulo 2 - Tratamentos dos Dados

2.1 - Conceitos Básicos	12
2.2 - Sistema Gerenciador de Bancos de Dados (SGBD)	14
2.3 - Regras Básicas para um SGBD	16
2.4 - Características Gerais para um SGBD	16
2.5 - Tipos de Bancos de Dados	18
2.5.1 - Bancos de Dados Hierárquicos	18
2.5.2 - Bancos em Rede	19
2.5.3 - Bancos Orientados a Objetos	20
2.5.4 - Bancos Relacionais	21
2.6 - O Modelo Entidade-Relacionamento (MER)	22
2.7 - Elementos de um MER	24
2.7.1 - Entidades	24
2.7.2 - Atributos	26
2.7.3 - Relacionamentos	26
2.7.3.1 - Relacionamento Unário	27
2.7.3.2 - Relacionamento Binário	28
2.7.3.3 - Relacionamento Ternário	28
2.7.4 - Cardinalidades	29
2.8 - Elementos de um Banco de Dados	30
2.8.1 - Tabelas	31
2.8.2 - Chaves	31

Capítulo 3 - Normalização

3.1 - Conceitos Básicos	34
3.2 - Primeira Forma Normal (1FN)	34
3.3 - Segunda Forma Normal (2FN)	37
3.4 - Terceira Forma Normal (3FN)	38
3.5 - Quarta Forma Normal (4FN)	39
3.6 - Quinta Forma Normal (5FN)	40
3.7 - Algoritmos práticos para aplicar as três primeiras formas normais	41

3.7.1 - Para passar à 1FN.....	41
3.7.2 - Para passar à 2FN.....	42
3.7.3 - Para passar à 3FN.....	43

Capítulo 4 - Linguagem de consulta Estruturada (SQL)

4.1 - Conceitos Básicos.....	45
4.2 - Subdivisões da SQL: DDL e DML	46
4.3 - Principais Comandos	47
4.3.1 - Comandos <i>Create Database/CreateTable</i>	47
4.3.2 - Comando <i>Open</i>	48
4.3.3 - Comando <i>Drop</i>	48
4.3.4 - Comando <i>Alter Table</i>	48
4.3.5 - Comando <i>Select</i>	49
4.4 - Equi-Junção	49
4.5 - Auto Relacionamento	50
4.6 - Sub-consultas	51
4.7 - Uniões.....	51
4.8 - Inserções, Atualizações e Exclusões	52
4.8.1 - Inserir (<i>Insert</i>).....	52
4.8.2 - Atualizar (<i>Update</i>)	52
4.8.3 - Excluir (<i>Delete</i>).....	52
4.9 - Transações.....	52
4.10- Visões	53
4.11- Exemplos de instruções sql.....	53
4.12- Exercício proposto	61

Capítulo 5 - Servidores de Bancos de Dados

5.1 - Introdução	62
5.2 - Firebird	62
5.2.1 - <i>Download</i> e Instalação	62
5.2.2 - Trabalhando com o servidor Firebird.....	64
5.3 - A Ferramenta IBExpert	77
5.3.1 - Criando um banco de dados no servidor local	80
5.4 - Criação de tabelas com o IBExpert	85
5.5 - Criação de chaves estrangeiras com o IBExpert	90
5.6 - Criação de chaves secundárias com o IBExpert	96
5.7 - Trabalhando com Domínios	99

5.7.1 - Criação de Domínios manualmente.....	99
5.7.2 - Criação de Domínios interativamente	103
5.7.3 - Testando o domínio criado	106
5.8 - Trabalhando com <i>Triggers</i>	110
5.9 - Entrando com dados nas tabelas	116
5.10- Tipos de dados suportados pelo Firebird	120
5.11- Trabalhando com <i>Generators</i>	121
5.12- <i>Views</i> (visões)	126
5.13- Trabalhando com <i>Stored Procedures</i>	128
5.14- Execução de instruções sql no Firebird com o IBExpert.....	135
5.14.1 - Inserção de registros em tabelas	135
5.14.2 - Criação de tabelas	140
5.15- Trabalhando com campos BLOB.....	148
5.16- Segurança no Firebird	153
5.16.1 - Criando novos usuários para o servidor.....	153
5.16.2 - Alterando a senha padrão do servidor Firebird.....	158
 Roteiro de Práticas	 160
 Bibliografia e Referências Bibliográficas	 166

Capítulo 1 - Sobre Tecnologia da Informação

1.1 - Introdução

Segundo O'Brien (2003) “dados são fatos ou observações crus, normalmente sobre fenômenos físicos ou transações de negócios”. Objetivamente, um dado pode ser considerado *qualquer coisa sobre a qual se tem algum interesse*. O que se deseja é manipular (processar) os dados para obter conhecimento necessário com a informação, e tomar as decisões mais corretas para a organização. Podemos, por assim dizer, que o(s) dado(s) são informação(s) em estado bruto, e a informação são esses dados processados através de procedimentos adequados. O esquema da **Figura 1.1** ilustra a situação clássica do ponto de vista teórico, como é um processo de processamento de dados em um sistema aberto.

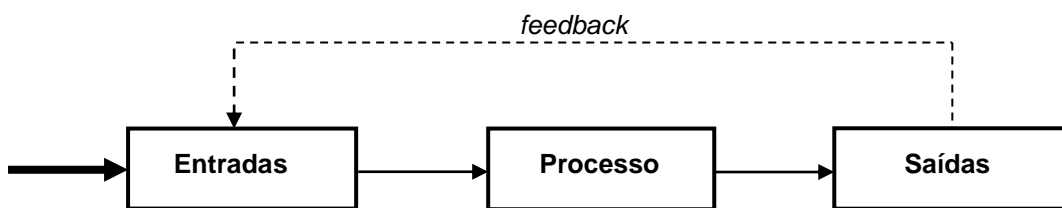


Figura 1.1 - Esquema clássico de um sistema aberto

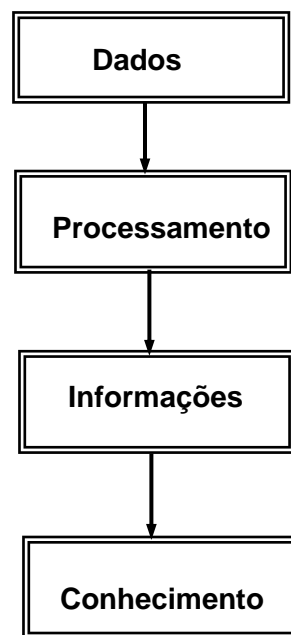


Figura 1.2 - Manipulando dados para obter conhecimento

Fonte: Criação de *modelos econométricos para medição do desempenho de uma empresa de pequeno porte*. In: XXII Encontro Nacional de Engenharia de Produção, 2002, Curitiba.

A **figura 1.2** mostra que o objetivo real do processamento de dados (em quaisquer situações) é o conhecimento, que é a forma ideal para que as decisões tomadas sejam as mais corretas possíveis para uma organização. A **figura 1.2** mostra como as práticas dentro de uma organização depende de três parâmetros básicos.

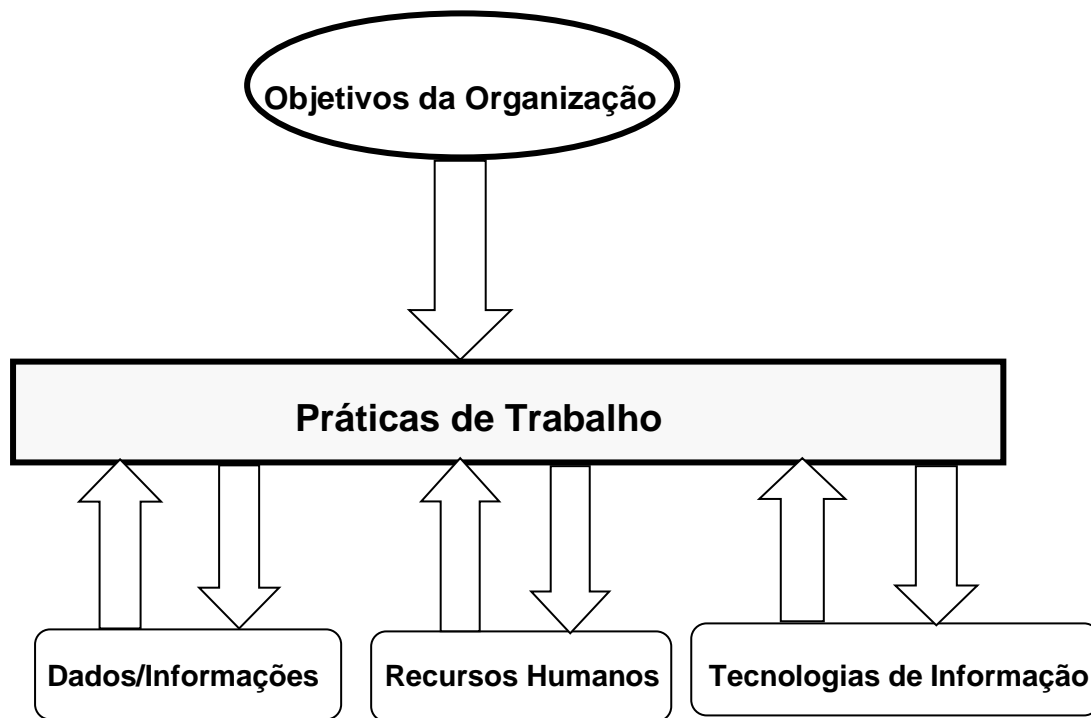


Figura 1.2 - Um Sistema de Informações Gerenciais no âmbito da organização

Fonte: CAMPOS FILHO, Maurício Prates de – “Os Sistemas de Informação e as modernas tendências da tecnologia e dos Negócios” - São Paulo, RAE Dez/1994 .

O esquema da **figura 1.3** mostra que um Sistema de Informações (SI) é composto, basicamente, de Tecnologias da Informação (TI's) e de Recursos Humanos (RH). Esses parâmetros que permitem a gestão da organização, através do controle da informação, independente do seu porte.

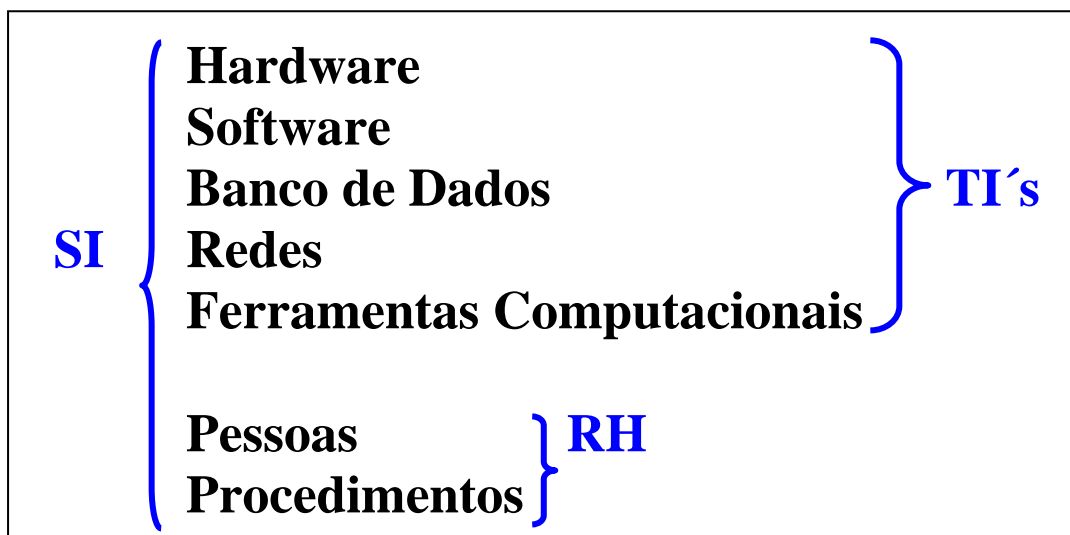


Figura 1.3 - Tomadas de decisão dependem das tecnologias de informação

Desse modo, pode ser observado que o controle, processamento e disseminação da informação são pontos fundamentais na gestão empresarial. Entretanto, é importante enfatizar: para processar corretamente os dados e obter informações gerenciais é necessário, antes, que esses dados estejam adequadamente armazenados. Então, a

questão seguinte a ser examinada é: *como armazenar os dados e/ou informações obtidas de maneira correta e confiável?*

Existem vários modos de armazenar dados/informações, mas todos eles convergem para um só: o uso de arquivos. Isto porque, que pela própria definição: *“arquivo é um local onde são guardados os dados/informações”*. Existem vários tipos de arquivos: arquivos-texto (puros, do tipo **txt** ou formatados **doc** do MS-Word), arquivos executáveis (**exe** ou **bat**), arquivos de figuras (**bmp**, **.gif**, **jpg**, **dib**), arquivos-fonte de programas (**pas**, **prg**, **c**, **vbp**, **for**, **asm**, ...) do tipo **“doc”** (para editores de texto) ou mesmo em arquivos do tipo **“dat”** (arquivos de dados gerais). Todos esses são tipos de arquivos, e por conseguinte poderiam, teoricamente, armazenar dados e ou informações. Mas, como processar e cruzar os diversos dados para obter as informações gerenciais necessárias às tomadas de decisão? Por exemplo, como obter informações a respeito das notas de determinado grupo de alunos de uma faculdade em cada disciplina, e classificadas em ordem decrescente? Outro exemplo, como listar os clientes de uma empresa com suas respectivas vendas e vendedores num determinado período, para ver se é importante enviar uma cartinha de felicitações com algumas promoções oferecidas pela empresa? Essas informações não podem ser obtidas empregando arquivos comuns para armazenar dados; é necessário um tipo de arquivo que armazene os dados de forma mais organizada e que propicie o cruzamento e consultas mais ágeis. Em outras palavras, é necessário Um Sistema de Informações bem elaborado que seja estabelecido em função da gestão da organização, e manipulado por Tecnologias de Informações (TI) adaptadas à estrutura organizacional.

A partir do início da década de 70 os volumes de dados e informações aumentaram muito em função das necessidades cada vez mais crescentes no âmbito das organizações. E a partir da década de 80 os Sistemas de Informação - mesmo ainda incipientes - já começavam a despontar como a solução para organizar melhor os dados nas empresas com o objetivo de gerar informações gerenciais. Com o fenômeno da globalização, a necessidade de se obter informações mais rápidas e mais confiáveis tornou-se um imperativo; uma questão de vida ou morte para as empresas. Por isso, o armazenamento, o cruzamento e o processamento de dados para se obter níveis cada vez melhores de informações passaram a ser fundamentais nos Sistemas de Informação, dentro do planejamento estratégico das organizações. Poloni (2001, p30) diz que: *“sistema de informação é qualquer sistema usado para prover informações (incluindo seu processamento), qualquer que seja sua utilização”*. Desse modo, tecnologia da Informação seria o veículo que difunde e irradia as informações de maneira rápida e eficiente. Então uma TI é o conjunto de *software/hardware* dos SI's, por isso, uma simples planilha eletrônica pode ser considerada uma TI, pois auxilia nas tomadas de decisão. Por outro lado, mesmo sendo considerada uma TI, uma planilha eletrônica não satisfaz plenamente o gerente tomador de decisão, pois apesar de exibir a informação deseja, não tem a capacidade de armazenar os dados de maneira conveniente e cruzá-los de modo a permitir informações de vários locais da empresa. O mesmo acontece com os *softwares* de computação numérica como MatLab, SciLab, Octave, etc, pois apesar de auxiliarem nas tomadas de decisão (baseadas em resultados numéricos), não se constituem em arquivos de dados possíveis de serem processados.

1.2 - Recursos e Tecnologias dos Sistemas de Informação

Um sistema de informação baseado em computador (SIBC) é composto pelo *hardware*, *software*, banco de dados, telecomunicações, pessoas e procedimentos, que estão configurados para coletar, manipular, armazenar e processar dados em informação (e até vice-versa, conforme a situação que se apresenta).

A **figura 1.4** ilustra, esquematicamente, como os sistemas de informação das empresas devem ser planejados, de modo a fornecerem informações gerenciais para as tomadas de decisão; ela mostra como um SI é composto: Recursos Humanos, Recursos de *Software*, Recursos de *Hardware*, Recursos de Rede e Recursos de Dados.

Objetivamente, um SI pode ser entendido como sendo a composição de dois componentes:

- Recursos Humanos
- Tecnologias da Informação (TI's)

As TI's são baseadas em Recursos de Software, Recursos de Hardware e Recursos de Dados. Este último componente (Recursos de Dados), talvez seja o mais importante, na medida em que através dele é que são tomadas as decisões nas empresas, baseada nas informações resultantes do tratamento adequado dos dados armazenados em repositórios.

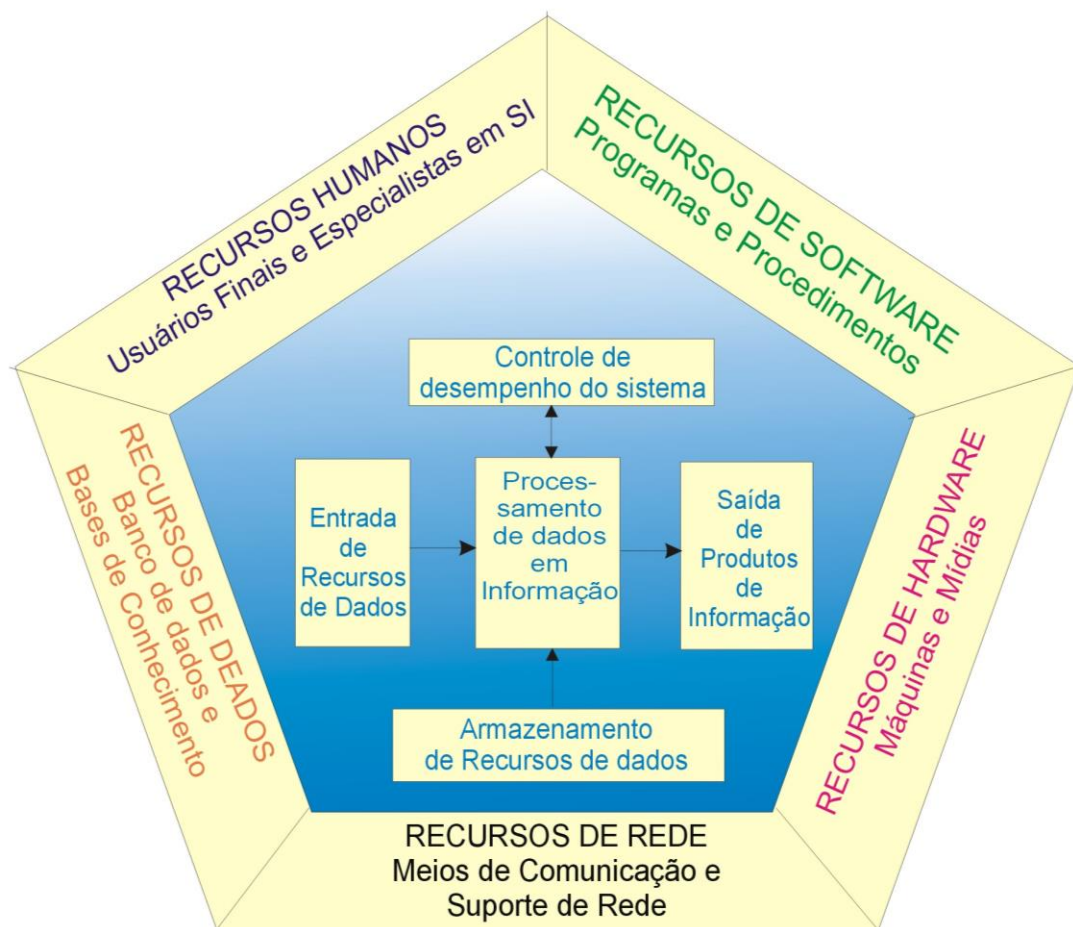


Figura 1.4 - Os componentes de um Sistema de Informação
 Fonte: Internet

Recursos de Hardware: consiste do computador usado para executar as atividades de entrada, processamento e saída. Os dispositivos de entrada incluem o teclado, os dispositivos de escaneamento automático, equipamento de caracteres em tinta magnética e muitos outros dispositivos. Os dispositivos de processamento incluem unidade central de processamento, memória e armazenagem. Existem muitos dispositivos de saída, incluindo as impressoras e os monitores de vídeo, os quais apresentam as telas do sistema.

Recursos de Software: consistem nos programas e nas instruções dadas ao computador e ao usuário. Esses programas e instruções permitem ao computador processar folhas de pagamento, enviar faturas aos clientes e fornecer aos administradores, informações para aumentar os lucros, reduzir os custos e proporcionar um serviço melhor ao cliente.

Recursos de Armazenamento: é uma coleção organizada de fatos e informações. O banco de dados de uma empresa pode conter fatos e informações sobre clientes, empregados, estoque, informações sobre vendas de concorrentes. Banco de dados é uma das partes mais valiosas e importantes de um sistema de informação baseado em computador.

Recursos de rede: permitem as empresas ligarem os computadores em verdadeiras redes de trabalho. As redes podem conectar computadores e equipamentos de computadores em um prédio, num país inteiro ou no mundo. As redes internas são exemplos mais comuns, além, obviamente da rede mundial: Internet.

Recursos Humanos: elemento mais importante na maior parte dos sistemas de informação baseado em computador. Os profissionais de sistemas de informação incluem todas as pessoas que gerenciam, executam, programam e mantêm o sistema computacional. Os usuários são administradores, tomadores de decisões, empregados e outros que utilizam o computador em seu benefício. Alguns usuários de computador também são profissionais de SI.

Procedimentos: incluem as estratégias, políticas, métodos e regras usadas pelo homem para operar o SIBC. Por exemplo, alguns procedimentos descrevem quando cada programa deve ser rodado ou executado. Outros procedimentos descrevem quem pode ter acesso a certas informações em um banco de dados.

Capítulo 2 - Tratamentos dos Dados

2.1 - Conceitos Básicos

Para os gerentes que tomam as decisões, a fonte dos dados tratados é muito importante, uma vez que precisam confiar nas informações dali extraídas. Por isto as bases de dados se destacam no contexto das TI's, de acordo com a estratégia organizacional. Atualmente existem gigantescas bases de dados, tratando e gerenciando dados e informações em todos os setores da sociedade moderna. A lista telefônica é um exemplo clássico, assim como os catálogos com produtos que são oferecidos via Internet, informações sigilosas sobre as grandes corporações, ou mesmo dados confidenciais sobre os cidadãos de um país, lembrando muito bem a filosofia do "Grande Irmão" contada por George Orwell no seu livro "1984". Essas informações estão guardadas em verdadeiros armazéns de dados, os chamados "Data Warehouse"; movimentações bancárias do dia a dia dos clientes dos bancos comerciais são gerenciadas por poderosos bancos de dados à prova de falhas. E como foi dito anteriormente, o armazenamento de dados, sejam volumes pequenos ou grandes, sempre requerem tratamentos na obtenção de informações para se tomar alguma decisão; armazenar dados apenas por armazenar não faz o menor sentido prático. Deste modo, os bancos de dados são ferramentas fundamentais nos dias de hoje, onde a informação se tornou a mercadoria mais valiosa para as organizações. Mas, um banco de dados não pode ser comparado a uma simples tabela com dados dispostos em colunas e linhas; é muito mais do que isto. Um arquivo contendo uma série de dados de um cliente como, por exemplo, arquivos no formato **dbf** (xBase) mesmo tendo alguma relação definida entre eles, não pode ser considerada um banco de dados real, pois é necessário bem mais funcionalidades para gerenciar esses dados de maneira correta e objetiva.

De um modo geral um banco de dados é definido como sendo "*o conjunto de dados e informações de uma organização*". Entretanto, é importante salientar que um banco de dados contém os dados dispostos numa ordem bem determinada em função do projeto do sistema, e obedecendo à estratégia da organização, com objetivos bem definidos e representando aspectos do mundo real. Assim sendo, uma base de dados (ou banco de dados, ou BD – ou ainda, *data base* em Inglês) é uma fonte de onde poderemos extrair uma gama de informações derivadas (gerenciais), com um nível de interação com eventos do mundo real que representa. Uma forma muito comum de se interagir com um banco de dados é através de sistemas específicos, normalmente aplicações desenvolvidas com ferramentas RAD (*Rapid Application Development*) que acessam as informações geralmente através de linguagens de 4ª Geração, e em particular a linguagem SQL, numa plataforma que utilize interface gráfica. Delphi e VB são exemplos dessas ferramentas.

A **figura 2.1** mostra um esquema de como os usuários de um banco de dados (profissionais e usuários finais) interagem com o banco.

Os **Administradores** de banco de dados (DBA) são responsáveis pelo controle ao acesso aos dados e pela coordenação da utilização do BD. São os profissionais mais requisitados para resolver problemas operacionais relacionados ao banco.

Os **Projetistas** de banco de dados (DBP) são os profissionais que identificam os dados a serem armazenados, a forma como devem ser representados, e influenciam na escolha do formato mais adequado para a empresa.

Os **Analistas** desenvolvem a lógica de como os dados podem ser relacionados e as dependências entre eles, através de modelos.

Os **Programadores** criam programas que acessam os dados da forma mais ágil e mais segura para obter as informações gerenciais nas tomadas de decisão.

O **Usuário** final é quem interage diretamente com o banco de dados, e deve sempre fornecer o *feedback* aos programadores/analistas para eventuais alterações no banco no sentido de melhorar a performance geral do sistema.

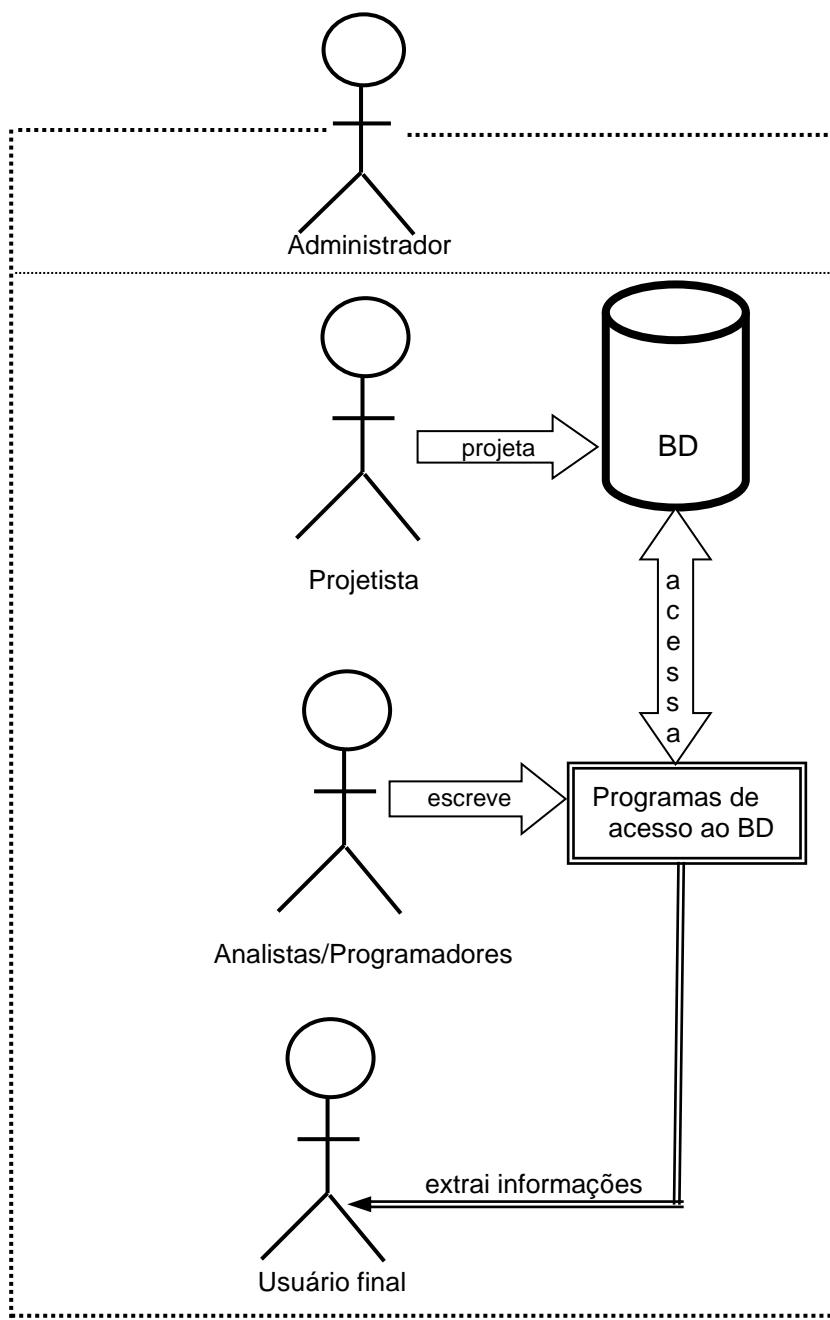


Figura 2.1 - Os profissionais que usam o banco de dados

Fonte: "Acessando Bancos de Dados com Ferramentas RAD: Aplicações em Delphi"
Rio de Janeiro, Brasport, 2008 (do autor).

Fisicamente, um BD pode ser descrito por três camadas, que forma a sua arquitetura:

- Núcleo Interno (estrutura e armazenamento físico)
- Núcleo Intermediário (descrição lógica dos dados)
- Núcleo Externo (visão oferecida ao usuário)

A **figura 2.2** mostra, esquematicamente, a arquitetura de um banco de dados.

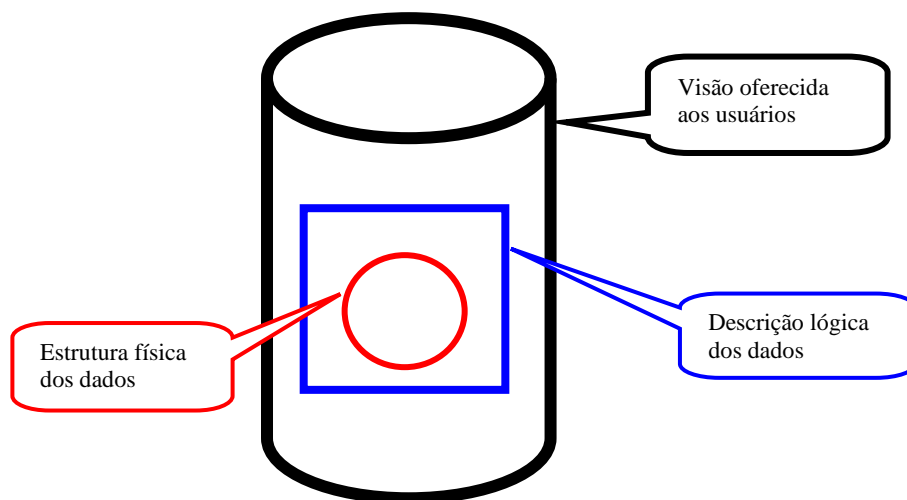


Figura 2.2 - Arquitetura “física” de um banco de dados

Fonte: “Acessando Bancos de Dados com Ferramentas RAD: Aplicações em Delphi”
Rio de Janeiro, Brasport, 2008 (do autor).

2.2 - Sistema Gerenciadores de Banco de Dados (SGBD)^[1]

Segundo Turban *et al* (2003) “O programa de software (ou grupo de programas) que fornece acesso a um banco de dados é conhecido como Sistema Gerenciador de Banco de Dados”. Esta definição enfatiza o fato de um SGBD ser um *software* que permite acessar uma base de dados de maneira interativa e mais profunda. Outra definição mais detalhada é dada por O’Brien (2003): “Um sistema de gerenciamento de bancos de dados (DBMS em inglês) é um conjunto de programas de computador que controla a criação, manutenção e uso dos bancos de dados para uma organização e seus usuários finais”. Ou de maneira mais técnica: “uma coleção de dados inter-relacionados, representando informações sobre um domínio específico”.

De acordo com O’Brien (2003) os dados estão armazenados no banco em “locais” bem determinados para uso dos tomadores de decisão na organização. Detalhando melhor, esses “locais” são, na verdade, as tabelas, que por sua vez se relacionam de acordo com especificações do analista e das necessidades dos usuários finais do sistema.

¹ Alguns autores usam a sigla SGBDR - **S**istema **G**erenciador de Banco de **D**ados **R**elacional -, para enfatizar que os dados manipulados se relacionam através de regras prescritas por modelos lógicos. Entretanto, aqui nesta apostila usaremos o termo SGBD.

A **figura 2.3** mostra um exemplo “físico” de base de dados, apresentando as tabelas que armazenam os dados relativos a uma situação ligada à área comercial.

- **Cliente** Armazena os dados pessoais dos clientes;
- **Venda** Armazena os dados relativos às vendas feitas aos clientes;
- **ContaReceb** Armazena os dados relativos às contas a receber dos clientes;
- **Vendedor** Armazena os dados pessoais dos vendedores da empresa;
- **TipoVenda** Armazena os dados relativos aos tipos de vendas (à vista, à prazo, etc);
- **Cidade** Armazena os dados relativos a cidades;.
- **Estado** Armazena os dados relativos aos estados;
- **Visão** Exibe informações oriundas do cruzamento de dados de tabelas.

Estas tabelas compõem um único arquivo (**Empresa.xxx**); este arquivo é o banco de dados, o qual é composto de várias tabelas, e mais alguns outros elementos oriundos dos dados das tabelas.

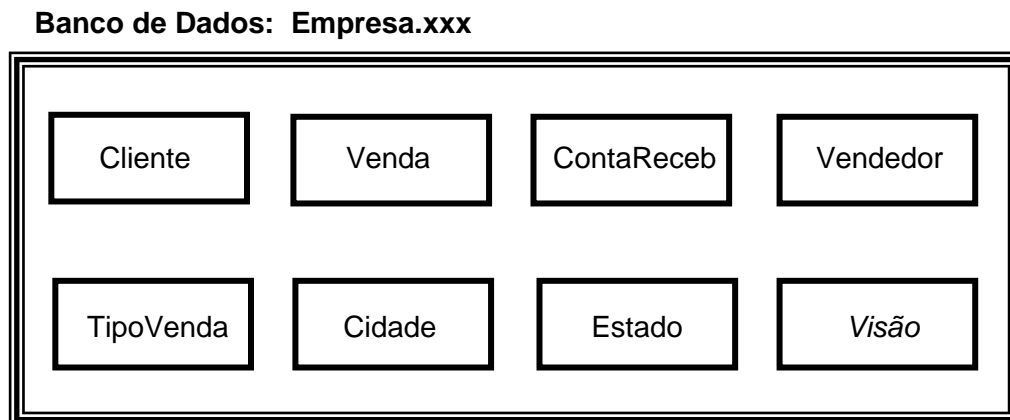


Figura 2.3 - Esquema “físico” de um banco de dados

A **figura 2.3** mostra um esquema de banco de dados (arquivo **Empresa.xxx**) com sete tabelas que armazenam dados, onde a extensão “xxx” dá o tipo de formato do banco de dados para aqueles com formato de único arquivo; por exemplo, **mdb** (para o Access), **gdb** (para o Interbase), **fdb** (para o Firebird), **mdf** (para o SQL Server), **ora** (para o Oracle) e assim por diante. E a “tabela” *Visão*, o que ela armazena?! Na verdade *Visão* não seria uma tabela propriamente dita (não contém dados), mas o resultado do cruzamento (relacionamento) dos dados de outras tabelas com informações para tomadas de decisão; por exemplo, poderia ser a relação de todas as vendas de clientes que compraram acima de determinado valor. Assim, o gerente de vendas poderia tomar a decisão de oferecer algum tipo de vantagem comercial a esses clientes, mantendo-os fiéis à empresa. Um SGBD pode fazer isso: cruzar dados de várias tabelas, manipulando os dados armazenados nas diferentes tabelas, através de modelos bem determinados e apresentar o resultado na forma de visões (consultas).

2.3 - Regras Básicas para um SGBD

Um SGBD “puro sangue” deve seguir algumas regras, que estão resumidas na **tabela 2.1**. Se pelo menos uma dessas regras for quebrada, o sistema pode ser considerado apenas um GA (gerenciador de arquivos) qualquer, mas não um perfeito SGBD.

Regra	Descrição
Auto-Contenção	Um SGBD não deve conter apenas dados, mas armazenar suas descrições, relacionamentos e formas de acesso: os <i>metadados</i> . A forma de acesso aos dados também deve ser definida pelo próprio sistema gerenciador, e não pela aplicação que o acessa.
Independência dos Dados	Um SGBD deve poder promover mudanças na estrutura do banco de dados sem precisar de intervenção direta da aplicação. Nenhuma definição dos dados pode ficar a cargo da aplicação.
Abstração dos Dados	Um SGBD deve fornecer ao usuário final apenas aquilo que lhe é necessário: a representação conceitual dos dados, e nunca os detalhes de como os dados estão relacionados. O modelo de dados é uma abstração que fornece ao usuário final essa representação. A criação e manutenção de chaves e índices devem ser transparentes ao usuário, de modo que as validações de entradas no banco devem ser feitas pelo sistema gerenciador e não deve ser visível externamente.
Visões	Um SGBD deve permitir que o usuário final tenha visões dos dados diferentes daquelas armazenadas originalmente no banco; e não deve haver replicações da estrutura para se conseguir essas visões.
Transações	Um SGBD deve ser capaz de gerenciar a integridade dos dados, sem o auxílio da aplicação. Ele deve conter instruções que permitam gravar várias alterações simultâneas, além de permitir que sejam canceladas quando isto for necessário, com apenas uma única instrução.
Acesso Automático	Um SGBD deve ser capaz de evitar o travamento de simultâneo de registros relacionados, quando ocorre o chamado <i>dead lock</i> . Esta ação de salvaguarda deve ser executada pelo SGBD, e não pela aplicação.

Tabela 2.1 – Regras de um Sistema Gerenciador de Banco de Dados

Caso alguma das regras estabelecidas na **tabela 2.1** não for satisfeita, o sistema que controla o banco de dados pode ser considerado apenas Gerenciador de Arquivos (GA), mas não um Sistema Gerenciador de Banco de Dados.

2.4 - Características Gerais de um SGBD

Os Sistemas Gerenciadores de Banco de Dados possuem seis características operacionais que sempre devem ser observadas. A **tabela 2.2** resume essas características.

Característica	Descrição
Controle de Redundâncias	Um SGBD não deve permitir redundâncias (duplicação) desnecessárias de dados. O armazenamento de determinada informação deve aparecer em um único local.
Compartilhamento dos Dados	Um SGBD deve possuir <i>software</i> de controle de concorrência ao acesso dos dados de modo a garantir escrita e leitura corretas dos dados em quaisquer situações.
Controle de Acesso	Um SGBD deve possuir meios de garantir o acesso ao banco somente aos usuários autorizados, de acordo com seu nível de prioridade.
Interfaceamento	Um SGBD deve oferecer formas de acesso ao dados, não apenas em linha de comando, mas também através de interfaces gráficas, permitindo uma maior interatividade com o usuário.
Esquematisação	Um SGBD deve fornecer mecanismos que possibilitem uma boa compreensão do relacionamento entre as tabelas do banco, assim como dar-lhes manutenção, quando for necessário.
Backups	Um SGBD deve oferecer ferramentas para se obter cópias do banco de dados, e também recuperar essa cópia quando for preciso, com o mínimo de ajuda externa.

Tabela 2.2 - Principais características de um SGBD

A **tabela 2.2** mostra as principais características desejáveis que um SGBD deve ter; entretanto, na prática, sempre há a possibilidade de se encontrar sistemas de gerenciamento que não satisfazem completamente todas essas seis características; mas isto não condena esses tipos de gerenciadores. O Access é um caso típico de sistema gerenciador que se encaixa perfeitamente neste perfil de *pseudo* SGBD, mas que atente perfeitamente a maioria das empresas - principalmente as micros e pequenas - dentro da realidade brasileira, a despeito de alguns profissionais que não o consideram um SGBD.

Podemos ter um banco de dados “modelo superior”, que respeita integralmente as regras básicas mostradas na **tabela 2.1** e que possua todas as características apresentadas na **tabela 2.2**, e por outro lado um outro “modelo inferior” que apesar de respeitar todas as regras básicas, não possua alguma característica desejável, mas tem um ótimo desempenho na empresa. Se em regime de trabalho o “modelo superior” não apresenta desempenho melhor que o “modelo inferior”, a aquisição do “modelo superior” deve ser questionável, pois o que interessa é o retorno da ferramenta no ambiente de trabalho, e não o fato de ser “superior” ou não! Como diria aquele velho dirigente chinês: “*não importa a cor do gato, o importante é que ele pegue ratos*”. Portanto, a escolha do banco de dados da empresa é muito importante, e a decisão por um ou outro modelo ou formato deve ser tomada por especialistas com grande experiência, mas afinado com as reais necessidades da organização, pois o investimento às vezes é alto e o desempenho do sistema deve ser o melhor possível.

2.5 - Tipos de Bancos de Dados

2.5.1 - Bancos de Dados Hierárquicos

Esses bancos seguem o estilo de um organograma empresarial (como por exemplo, Diretoria→Gerência→Departamento→Setor) ou então a organização de uma biblioteca (Informática→Programação→Delphi). Esse modelo é capaz de representar um tipo de organização de forma direta, mas apresenta um inconveniente quando esta situação não aparece claramente com relações de hierarquia. A **figura 2.4** mostra um exemplo de “Cliente”, para esclarecer melhor o estilo desse modelo.

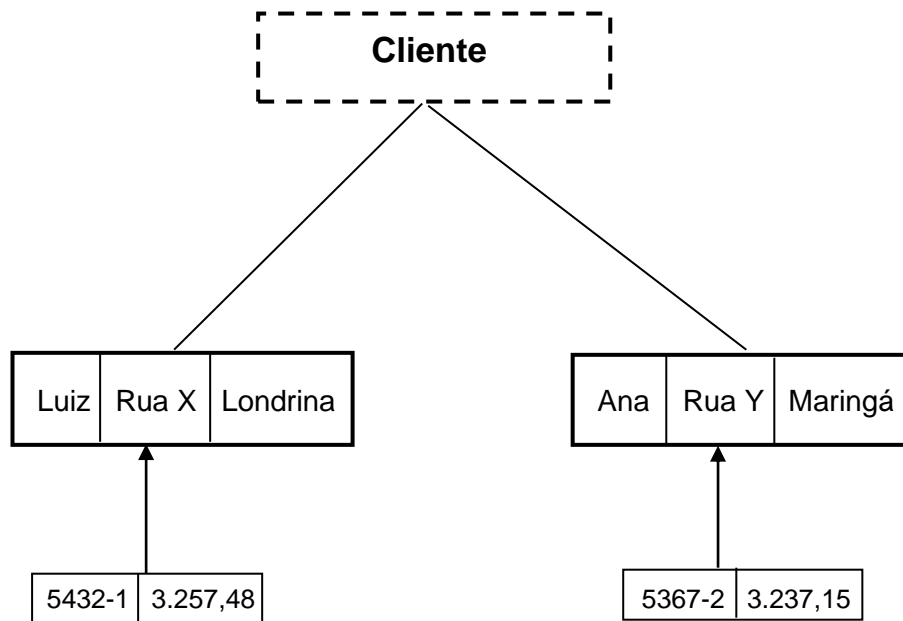


Figura 2.4 - Esquema de um Banco de Dados Hierárquico

Esse modelo de banco de dados foi o primeiro a ser desenvolvido, devido à consolidação dos discos de armazenamento, muito populares na época, e que exploravam essa a estrutura física desses bancos. Nesse modelo os dados são estruturados em hierarquias ou árvores. Os nós das hierarquias contêm ocorrências de registros, onde cada registro é uma coleção de campos (atributos), e cada um contendo apenas uma informação. O registro da hierarquia que precede a outros é o *registro-pai*, os outros são chamados de *registros-filhos*. Nesse tipo de banco os acessos são muito rápidos, pois suas unidades formam uma estrutura ligada, o que facilita a pesquisa. Algumas propriedades dos bancos hierárquicos merecem ser citadas:

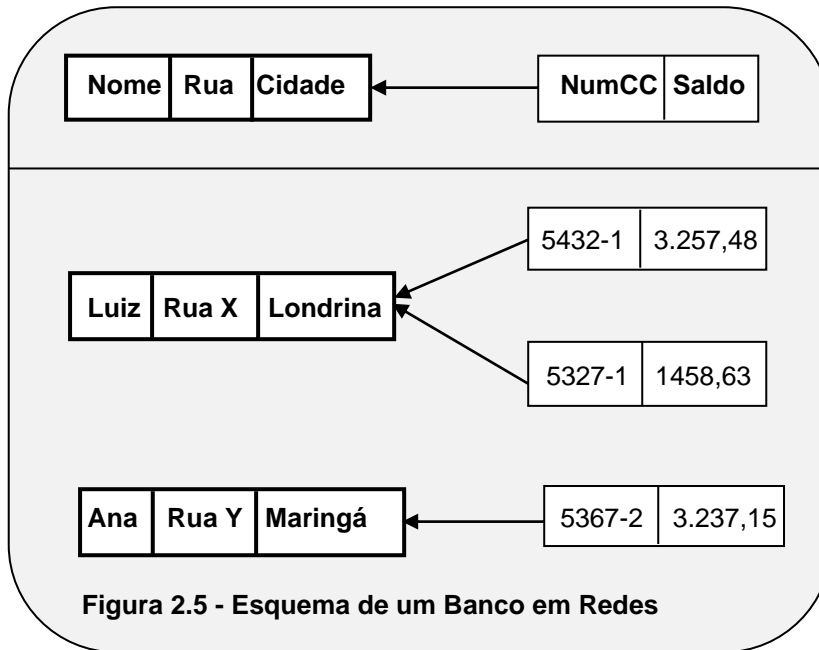
- ✓ O *registro-raiz* não participa de relacionamentos como *registro filho*;
- ✓ Cada tipo de registro, com exceção do *registro-raiz*, participa de apenas um relacionamento como registro filho.
- ✓ Um tipo de registro pode participar de qualquer número de relacionamentos como registro pai.
- ✓ Uma ligação é uma associação entre dois registros.
- ✓ vários *registros-filhos* possuem cardinalidade 1:N

A **figura 2.5** mostra um esquema-exemplo desse tipo de banco de dados; está indicado que os clientes têm: **nome**, **endereço** e **cidade**; além disso, possuem *conta corrente* com determinado *saldo* no momento em esquema de “árvore”, onde os dados pessoais do cliente estão num nível acima dos seus dados bancários.

O grande problema desse tipo de banco é que ocorrem muitas redundâncias nos dados, e a modelagem se torna muito complexa quando é preciso controlar itens e subitens dessas unidades; por exemplo, quando se depara com uma cardinalidade do tipo **N:M** (*muitos-para-muitos*), quando registros participam como filhos em mais de um relacionamento. Nesses casos evolui-se para o modelo em Rede. Um exemplo de banco de dados hierárquico é o ADABAS produzido pela empresa Software AG.

2.5.2 - Bancos de Dados em Rede

Este modelo é considerado uma extensão do modelo hierárquico para resolver o problema de relacionamentos com cardinalidade N:M entre os registros. No modelo em rede os registros são organizados em grafos onde aparece um único tipo de associação que define uma relação 1:N entre dois tipos de registros: *proprietário* e *membro*. Assim, dados dois relacionamentos 1:N entre os registros A e C e entre os registros C e D é possível construir um relacionamento M:N entre A e D. Neste modelo os dados são colocados em registros e classificados em classes que descrevem a estrutura de um determinado tipo de registro. Os registros são descritos em relações de conjuntos, o que permite estabelecer ligações lógicas entre eles. O esquema básico para esse tipo de banco é o conceito de ponteiro, onde um registro aponta para o local onde outro registro se encontra. No modelo de redes sempre existe um elemento distinto e a base que define a disposição de todos os outros. Desse modo, sempre existirá três elementos que define a estrutura dos bancos em rede: nome, tipo de *registro-base* e o tipo de *registro-filho*. Por exemplo, supondo uma base de dados que contenha uma tabela denominada “Disciplina” que dependa da tabela “Curso”; então, uma determinada disciplina seria um *registro-filho*, pertencente a um curso. Mas, o que restringe o modelo de Redes é, obviamente, o fato de que qualquer entrada estar sujeita a um determinado *registro-pai*; desse modo podemos concluir que cada ligação define um relacionamento *pai-filho*. E ao contrário do Modelo Hierárquico, em que qualquer acesso aos dados passa pela raiz, o modelo em rede possibilita acesso a qualquer nó da rede sem passar pela raiz. No modelo em rede o sistema comercial mais divulgado é o CA IDMS da *Computer Associates*. O diagrama para representar os conceitos do modelo em redes consiste em dois componentes básicos: *caixas*, que correspondem aos registros, e *linhas* que correspondem às associações. A **figura 2.5** ilustra um exemplo de esquema do modelo em rede, baseado numa extensão do exemplo apresentado na **figura 2.4**.



Por outro lado, é importante enfatizar que tanto o modelo hierárquico quanto o modelo em redes, são modelos orientados a registros; isto quer dizer que quaisquer acessos à base de dados para inserção, consulta, alteração ou remoção, são feitos em um registro de cada vez. Este não é o caso dos bancos de dados Orientados a Objeto, estudados (de maneira introdutória) a seguir...

2.5.3 - Bancos de Dados Orientado a Objetos

Este modelo é o mais “badalado” atualmente devido à publicidade (às vezes exagerada) de que ele é o “remédio para todos os males” em ambientes de desenvolvimento. Esse modelo começou a ser considerado viável a partir de meados dos anos 80 (século XX), motivado principalmente pela popularização dos sistemas CAD-CAM e dos sistemas de informações geográficas. Esse tipo de banco de dados é baseado nas características da tecnologia de orientação ao objeto. Neste caso, os dados são tratados como coleções de “coisas” que descrevem características do mundo real, baseados em objetos de classes previamente definidas. Por outro lado, apesar de ser o modelo “da onda”, os Sistemas Gerenciadores Bancos Orientados a Objetos (**OODBMS - Object-Oriented Database Manager System** - em inglês) ainda são alvo de estudos e pesquisas acadêmicas, pois sua aplicação direta na prática ainda provoca discussões, apesar de ser “chique” para um programador dizer que usa esse modelo. Os críticos do OODBMS argumentam que nesse modelo é difícil manipular dados muito complexos e, além disso, como o código é escrito em linguagem orientada a objetos (C++, Java, C#, VB.Net, Delphi, etc) ocorreria um descompasso entre o banco e o código de acesso a ele. O uso do OODBMS é dificultado na maneira de armazenar dados e operações (métodos) numa única estrutura tratada como um objeto, de modo a promover seu emprego realmente prático quando se trata de consultas gerais em ambientes comerciais. Um argumento (também discutível) dos críticos desse modelo, é que embora empregando análise Orientada a Objetos (com todos os aqueles conceitos: atores, diagrama de classes, diagrama de caso de uso, diagrama de seqüência, diagrama de pacote, diagrama de colaboração, etc), na hora de fazer os relacionamentos entre as entidades apelam para o mapeamento para o modelo Relacional, e acabam utilizando o MER (ou mapeando para o objeto relacional) para codificar os programas em vez de usar o Diagrama de Classes. Uma atitude não muito correta, mas é o que ocorre na grande maioria das vezes. De qualquer forma o Modelo Orientado a Objetos é um modelo altamente consistente e deve ser considerado na hora de optar por um determinado tipo de banco, observando a linguagem de programação.

Por tudo isso, hoje em dia é quase consenso de que os Bancos de Dados Orientados a Objetos serão usados em aplicações especializadas, deixando com os sistemas relacionais a aplicação prática nos negócios tradicionais, onde as estruturas de dados baseadas em relações mostraram que são suficientes para descrever os relacionamentos entre as entidades (tabelas). O diagrama mais importante empregado nesse modelo é o Diagrama de Classes, baseado na linguagem de modelagem UML (**Unified Modeling Language**), servindo como um modelo de dados orientado a objetos. Observe o exemplo esquemático da **Figura 2.6**, e compare as diferenças com os modelos apresentados anteriormente. Observe que “Nome”, “Rua” e “Cidade” são os atributos de um *cliente*; assim como “NumCC” e “Saldo” o são para *conta*, como nos casos anteriores. Entretanto, surgem duas novidades: primeiro é que *cliente* e *conta* agora se apresentam como entidades (classes) separadas, unidas por um traço que representa a associação entre elas. Outra novidade, que dificulta um pouco o emprego prático do modelo OO, é a inserção de operações na própria classe: os chamados métodos. Na **Figura 2.6** *Cadastrar()* é um dos métodos da classe “Cliente”, e *Atualizar()* método da classe “Conta”.

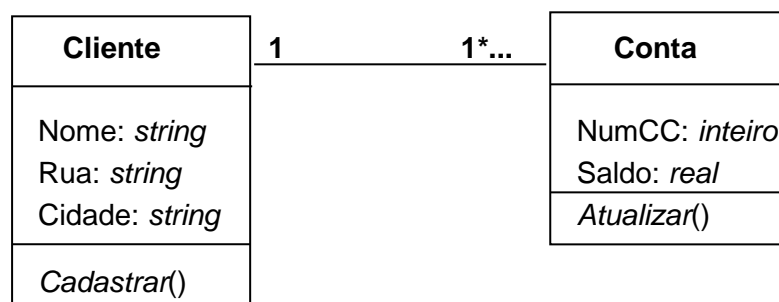


Figura 2.6 - Esquema de um Banco de Dados Orientado a Objetos

2.5.4 - Bancos de Dados Relacionais

O modelo relacional é um modelo baseado na matemática, e foi criado por Edgard Frank Codd - matemático e pesquisador da IBM - com o objetivo de descrever como as bases de dados devem funcionar, tendo como suporte a teoria dos conjuntos e álgebra relacional.

Este modelo revelou ser o mais flexível e adequado ao solucionar os vários problemas que se colocam no nível de concepção e implementação da base de dados. A estrutura fundamental do modelo relacional é a relação (tabela). Uma relação é constituída por um ou mais atributos (campos) que traduzem o tipo de dado a armazenar. Cada instância do esquema (linha) é chamada de *tupla* (registro). Os dados são representados por relações, onde cada relação, rigorosamente falando, é subconjunto do produto cartesiano de n conjuntos. Essas relações contêm informações sobre as entidades representadas e seus relacionamentos. Esse modelo é baseado no conceito de matrizes, onde as linhas representam os registros e as colunas os campos. Os nomes das relações (tabelas e o resultados dos seus relacionamentos) e dos campos são fundamentais para a compreensão entre o que está armazenado, onde está armazenando e qual a relação existente entre os dados armazenados. Cada linha de uma relação é chamada de TUPLA e cada coluna de ATRIBUTO. O conjunto de valores possíveis de serem assumidos por um atributo é chamado de DOMÍNIO. Do ponto de vista prático podemos ter as definições mostradas na **tabela 2.3**.

Análise de Sistemas	Programação	Processamento de Dados
Entidade	Tabela	Arquivo
Tupla	Linha	Registro
Atributo	Coluna	Campo

Tabela 2.3 - Comparação de termos empregados em Informática

Domínio é o conjunto de valores atômicos a partir dos qual um atributo retira seus valores. Portanto, “Rio de Janeiro”, “Paraná”, “Minas Gerais” e “Bahia” são estados (valores) válidos para o Brasil, enquanto que “Cochabamba” não é um estado brasileiro (é um departamento da Bolívia). O esquema de uma relação nada mais é do que os campos (colunas) existentes em uma tabela; já a instância da relação consiste no conjunto de valores que cada atributo pode assumir. Então, os dados armazenados no banco de dados são formados pelas instâncias das relações, e devem atender às seguintes condições:

- Não podem ser duplicadas (no conjunto de estados brasileiros não podem existir dois estados “Paraná”, por exemplo);
- A ordem de entrada de dados no banco não deverá ter qualquer importância para as relações;
- Os atributos devem ser atômicos, isto é, não podem definir uma relação dentro de outra.

Conforme já explicado, esse modelo ainda é o mais empregado na prática, pois é muito fácil de ser gerenciado, além de promover um perfeito relacionamento entre as tabelas que armazenam efetivamente os dados, e oferecer uma visão mais clara de como esses dados estão associados.

2.6 - O Modelo Entidade-Relacionamento

O modelo mais empregado para descrever como os dados estão armazenados num banco de dados relacional é o chamado **Modelo Entidade-Relacionamento (MER)**. O MER representa os dados agrupados em tabelas, e como se relacionam seguindo algumas regras especificadas *a priori*. Para que o processamento desses dados e a posterior informação sejam de qualidade é importante que se tenha uma boa modelagem, caso contrário o trabalho dos programadores será muito “pesado”, além de se ter uma manutenção do banco muito confusa e desgastante. O modelo relacional mostra explicitamente o mapeamento entre as diversas tabelas que compõem o banco de dados, e em nível de análise de sistemas essas tabelas são ditas “relações” ou “instâncias” do banco de dados, uma vez que podem existir fisicamente ou apenas logicamente.

Uma relação **R** pode ser representada pela seguinte notação tabular.

$$R = (\underline{A_1}, A_2, A_3, \dots, A_n)^{[2]}$$

R ==> identificador da relação

A_k (k=1,,n) ==> atributos da relação (os atributos sublinhados são chaves; no caso A1)

De uma maneira menos formal, podemos considerar uma relação como representante de uma tabela ou resultado do relacionamento (associação) entre tabelas.

² Uma relação no contexto de banco de dados é o resultado do produto cartesiano entre duas outras relações. Objetivamente, podemos considerar uma relação como sendo uma tabela ou o resultado do relacionamento entre tabelas, como acontece com as tabelas (entidades) associativas.

Na **figura 2.7** as associações que existem entre as tabelas são as seguintes:

- ✓ Um *cliente* pode se associar com muitas contas, mas uma *conta* só pode estar associada a apenas um único *cliente*;
- ✓ Um *cliente* pode se associar com muitas vendas, mas uma *venda* só pode estar associada a apenas um único *cliente*;
- ✓ Um *vendedor* pode se associar com muitas vendas, mas uma *venda* só pode se associar a um único *vendedor*.
- ✓ Uma *venda* pode estar associada a várias contas, mas uma *conta* só pode se associar à uma única *venda*.
- ✓ Um *tipo de venda* se associa a muitas vendas, mas uma *venda* só pode ser de um único *tipo*.

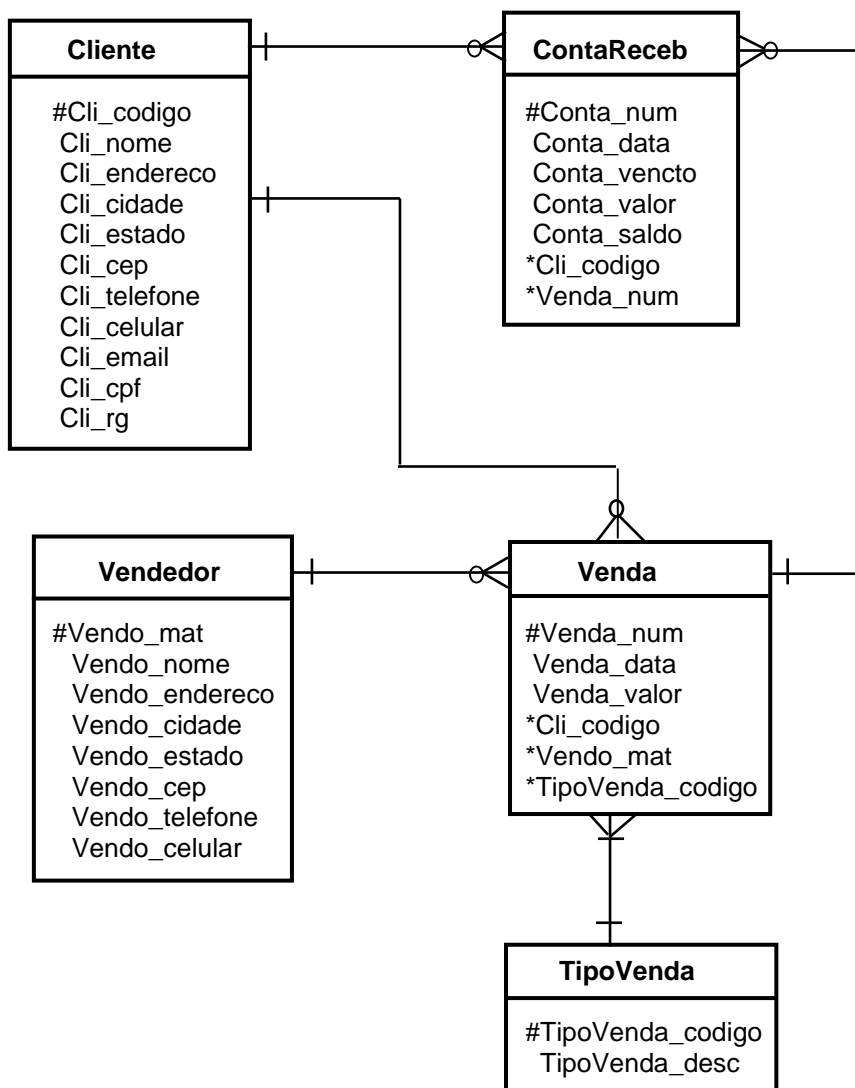


Figura 2.7 - Modelo de relacionamento entre as entidades do banco Empresa.mdb

Fonte: “Acessando Bancos de Dados com Ferramentas RAD: Aplicações em Delphi”
Rio de Janeiro, Brasport, 2008 (do autor).

O esquema da **figura 2.7** mostra as tabelas do banco de dados “Empresa” num modelo de dados denominado “Modelo Entidade-Relacionamento” (MER). Este é o modelo mais popular e o mais utilizado pelos analistas de sistemas para descrever logicamente o armazenamento de dados num banco relacional. Nesse modelo as tabelas são tratadas como entidades, ou mais formalmente, como classes de entidades.

2.7 - Elementos básicos de um MER

2.7.1 - Entidades

Como foi comentado, uma tabela também pode ser definida como uma entidade no ambiente de análise de sistemas; desse modo, uma entidade pode ser definida como: *“algo, concreto ou abstrato, de interesse na análise de um sistema, e que tenha necessidade de armazenar algum dado a seu respeito”*. Na **figura 2.7**, “Cliente”, “Venda”, “ContaReceb”, “Vendedor” e “TipoVenda” são exemplos de entidades. Objetivo do analista de sistemas é estabelecer o correto relacionamento entre as entidades de um MER para que o acesso aos dados seja eficiente e confiável. No modelo relacional, uma entidade é representada por um retângulo dividido em duas partes: na primeira sua identificação e na segunda parte seus atributos. Na parte de baixo do retângulo aparecem algumas “características” da entidade: essas “características” são os chamados atributos (na análise) e campos (na implementação). Os atributos “Cli_codigo”, “Conta_num”, “Venda_num”, “Vendo_mat” e “TipoVenda_codigo” são especiais: são chaves primárias das suas respectivas tabelas, sinalizadas com o caracter #. Os atributos marcados com asterisco (*) são as chaves estrangeiras. Por exemplo, **Luiz Carlos de Souza** poderia ser o valor do atributo “Cli_nome”, e representando uma entidade particular da classe-entidade “Cliente”, ou, mais formalmente uma instância da classe “Cliente”. O valor do seu atributo “Cli_codigo” poderia ser, por exemplo, **345** (o tri centésimo quadragésimo quinto cliente a ser cadastrado). Por outro lado, concordamos que **Luiz Carlos de Souza** é um nome bastante comum no Brasil; assim, em uma mesma tabela (classe-entidade) poderia existir um outro *Luiz Carlos de Souza*; mas certamente com outro código, diferente de 345. Esta é a restrição mais importante do modelo relacional: a não duplicidade da chave primária. E dependendo do contexto, uma entidade pode ser Forte, Fraca ou Associativa.

- **Entidades Fortes**

Uma entidade é dita forte quando existe por si só; sua existência não depende de qualquer uma outra para armazenar os dados. Desse modo, no MER da **figura 2.7** a entidade “Vendedor” é dita “entidade forte”, pois ela não depende de nenhuma outra entidade. Isto é, para cadastrar um vendedor não há necessidade de outra entidade existir. Na prática, considerando uma entidade é dita forte quando não contem o “pé de galinha” (conector que indica **muitos**).

- **Entidades Fracas**

Ao contrário das entidades fortes, uma entidade fraca não existe por si só; ela depende de uma outra para existir. Assim, no esquema da **figura 2.7** as entidades “Venda” e “ContaReceb” são exemplos desse tipo de entidade. No caso de “Venda”, para que uma *venda* ocorra são necessárias que existam três entidades simultaneamente: “Cliente”, “Vendedor” e “TipoVenda”. Em outras palavras, para que seja efetivada uma *venda* é necessário um cliente (que compra), um vendedor (agente da venda) e um tipo de venda para validar a transação. Assim também, para que uma *conta* seja cadastrada são necessários uma *venda* e um *cliente*.

- **Entidades Associativas**

Entidade associativa é um tipo de entidade que pode aparecer no contexto de criação

do banco de dados. Esse tipo de entidade depende de pelo menos duas outras entidades para existir, e só aparece em relacionamentos onde ocorre cardinalidade N:M. Na **Figura 2.7** não ocorrem esses casos, mas eles não são incomuns. Seja por exemplo, um caso em que exista um relacionamento entre “Medico” e “Paciente” (como mostrado na **Figura 2.8 a**). A associação entre um médico pode ter a seguinte semântica:

“Um médico pode atender a muitos pacientes, e um paciente também pode ser atendido por muitos médicos” ^[3].

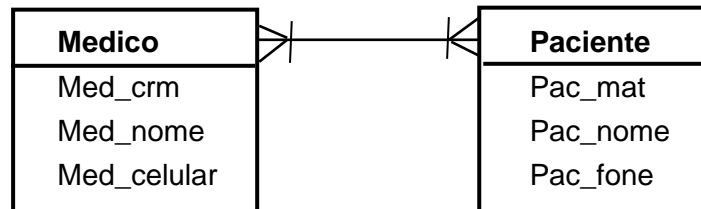


Figura 2.8a - Exemplo de relacionamento com cardinalidade N:M

A situação mostrada na **figura 2.8a** é indesejável do ponto de vista de futuras implementações para manipulação dos dados, pois o tomador de decisão certamente não desejará saber a respeito dos dados de um médico e nem de um paciente em particular; o que ele certamente desejará saber é, por exemplo, qual médico atendeu determinado paciente num determinado dia e o que foi tratado, quanto custou, etc. Nesses casos (de cardinalidades N:M) o que se faz é criar uma nova classe-entidade chamada de entidade do tipo “associativa”. No exemplo, essa nova classe-entidade dependerá, simultaneamente, de “Medico” e de “Paciente”, e terá cardinalidade N:1 nos dois sentidos. A **figura 2.8b** mostra um esquema de como a situação é resolvida.

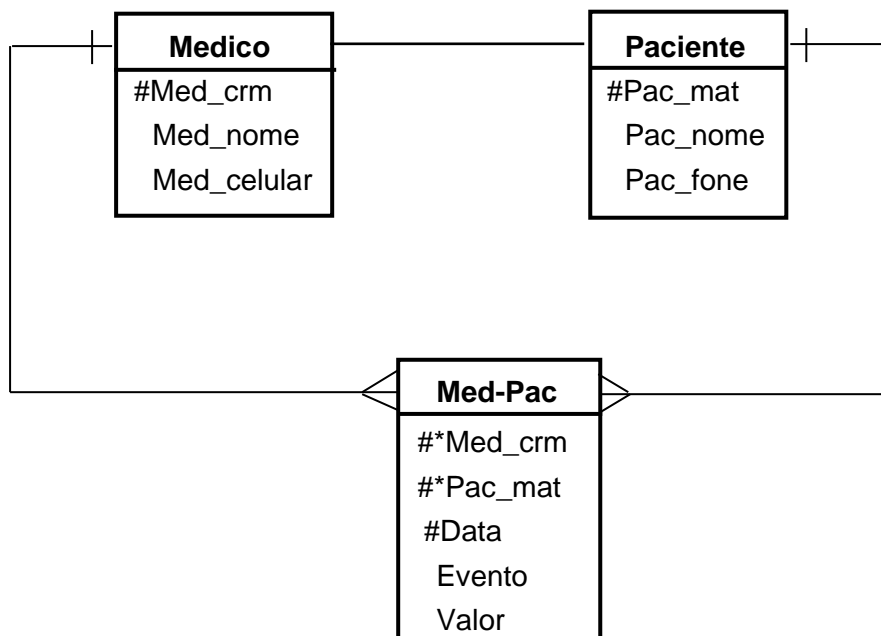


Figura 2.8b - Criando uma entidade Associativa

³ Observe que até agora todas as entidades foram nomeadas no singular (por exemplo, “**Venda**” em vez de “**Vendas**”). Isto se deve ao fato de que quando se fala em *entidade*, indicada por um retângulo com o nome na parte superior e os atributos na parte inferior, isto representa, na verdade, uma entidade em particular: uma venda. Por outro lado, podemos considerar isto como uma convenção; nada impede que o nome da entidade (aliás *classe-entidade*) seja nomeada no plural, o que normalmente é feito quando ela passa a ser *tabela* de um banco de dados.

Observe na **figura 2.8b** que a nova entidade (Med-Pac) “rouba” as chaves primárias das entidades das quais ela foi gerada: “Medico” e “Paciente”. Essas duas chaves (que agora passam a ser chamadas de “chaves estrangeiras”) são os atributos “Med_crm” (CRM do médico) e “Pac_mat” (matrícula do paciente). Os atributos “Data”, “Evento” e “Valor” são chamados de atributos de relacionamento, pois dependem do relacionamento entre as duas entidades progenitoras.

2.7.2 - Atributos

Conforme foi dito no item anterior, os atributos representam as características de uma entidade. Assim, para o MER da **figura 2.7** temos o seguinte para a tabela “Venda”:

- Venda_num Número de uma venda (*chave primária*);
- Venda_data Data da realização de uma venda;
- Venda_valor Valor na transação de uma venda;
- Cli_codigo Código do cliente envolvido numa venda (*chave estrangeira*);
- Vendo_mat Matrícula do vendedor que comandou uma venda (*chave estrangeira*);
- TipoVenda Código do tipo de venda (*chave estrangeira*).

A escolha dos atributos de uma entidade depende das necessidades dos usuários do banco, das informações que eles necessitam e para que querem determinados dados armazenados. Ao analista de sistemas cabe fazer as associações entre as entidades de maneira a se obter informações gerenciais. Por exemplo, será que o sexo do vendedor é importante? Então, nesse caso, seria necessário incluir um novo atributo: “Vendo_sexo”, que armazenaria o sexo do vendedor.

2.7.3 - Relacionamentos

Apesar de já termos colocado esse termo anteriormente, este item tratará sistematicamente das associações entre as entidades; associações essas, que no âmbito de um sistema de banco de dados são conhecidas, tecnicamente como “relacionamentos”.

A expressão “relacional” vem do fato de que as tabelas (entidades) que compõe o banco de dados se relacionarem através de seus atributos-chave, baseando na álgebra relacional. O mais importante na modelagem de dados (para qualquer modelo) é que as entidades se relacionem de maneira tal que a extração das informações seja mais rápida e mais confiável e sem dúvidas; afinal de contas é o que interessa ao tomador de decisões. De uma maneira menos formal, um relacionamento é uma “associação entre entidades”.

Afigura 2.9 explica, de uma maneira mais didática, um relacionamento com o auxílio da “Teoria dos Conjuntos”, mostrando que um *cliente* pode estar associado a uma ou várias contas, mas uma *conta* só pode se associar a um único *cliente*. Observem que o cliente **348** não está associado a nenhuma *conta*; isto pode acontecer, pois pode ser que esse cliente só compre e pague à vista; mas o contrário não pode: se existe uma conta a receber ela deve pertencer a algum cliente. Afinal de contas, quem pagará a *conta* se não tiver cliente?!

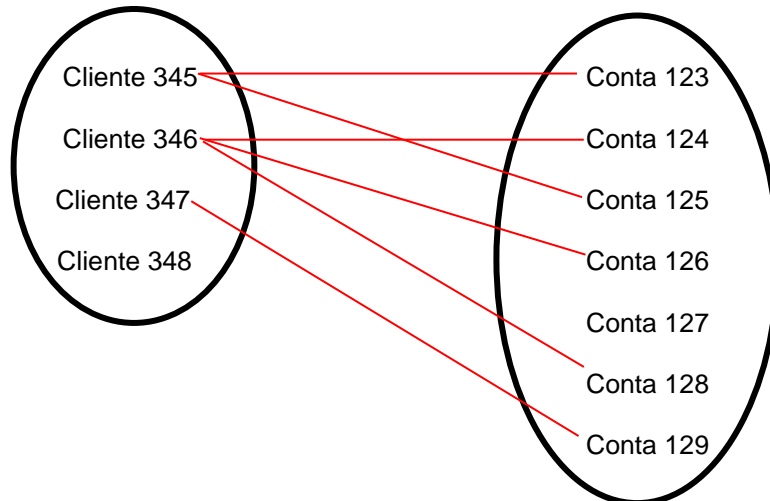


Figura 2.9 - Associações entre “Cliente” e “ContaReceb”

Quanto ao número de entidades envolvidas, podem existir (pelo menos na prática), três tipos de relacionamentos:

- Unário: uma entidade se relacionando com ela mesma;
- Binário: relacionamento entre duas entidades;
- Ternário: quando três entidades se relacionam simultaneamente.

2.7.3.1 - Relacionamento Unário

O relacionamento **Unário** é muito raro de acontecer; só em casos especiais. Nesse tipo de relacionamento apenas uma entidade participa (vide **Figura 2.10**); o relacionamento de uma entidade com ela mesma. Por exemplo, no contexto de um clube de lazer existe uma entidade denominada “Socios” que armazena os dados dos seus sócios (vide **Tabela 2.4**).

- Soc_cod armazena o código do sócio (chave primária);
- Soc_nome armazena o nome do sócio;
- Soc_sexo armazena o sexo do sócio;
- Soc_fone armazena o telefone do sócio;
- Soc_codprop armazena o código do sócio que o propôs ao clube.

Socios

Soc_cod	Soc_nome	Soc_sexo	Soc_fone	Soc_codprop
1234-05	Luciano Abrolho	M	9201-1144	2342-08
1245-05	Kátia Venâncio	F	9201-0035	2342-08
1342-06	Jacira Pimentel	F	9309-2345	3456-08
1453-06	Juliano Cabrera	M	9304-1092	3567-08

Tabela 2.4 - Tabela que armazena os dados dos sócios de um clube de lazer

Fonte: “Acessando Bancos de Dados com Ferramentas RAD: Aplicações em Delphi”
Rio de Janeiro, Brasport, 2008 (do autor).

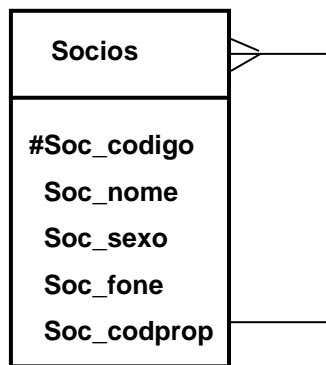


Figura 2.10 - Auto Relacionamento da entidade “Socios”

Uma das questões que sempre se coloca nesses casos de relacionamentos unários é a seguinte: “Como listar os nomes dos sócios com os nomes de seus respectivos sócios proponentes, sabendo que um sócio desta tabela também é um proponente?” Essa situação é oriunda do fato de um sócio ser proponente e proposto ao mesmo tempo. Isto caracteriza o chamado um auto relacionamento e a extração de informações é denominada *self join*.

2.7.3.2 - Relacionamento Binário

Este tipo de relacionamento é o mais comum no modelo relacional, onde participam bidirecionalmente, apenas duas entidades. A Figura 2.11 mostra o relacionamento entre as entidades “Medico” e “Especialidade”

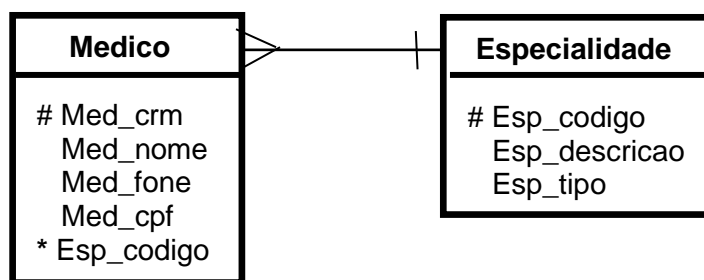


Figura 2.11 - Relacionamento entre “Medico” e “Especialidade”

2.7.3.3 - Relacionamento Ternário

Neste caso, são três as entidades que se relacionam simultaneamente; o exemplo mais comum é a clássica associação entre “Cliente”, “Conta” e “Agencia”, num sistema bancário. Apesar desse tipo de relacionamento ser possível (e existir) ele deve ser evitado na prática, devido à complexidade envolvida no seu tratamento. Caso ele ocorrer é aconselhável substituí-lo por relacionamentos binários, pois sua abordagem algorítmica nos programas é bastante complicada. A **figura 2.12** mostra o esquema relacional entre essas três entidades.

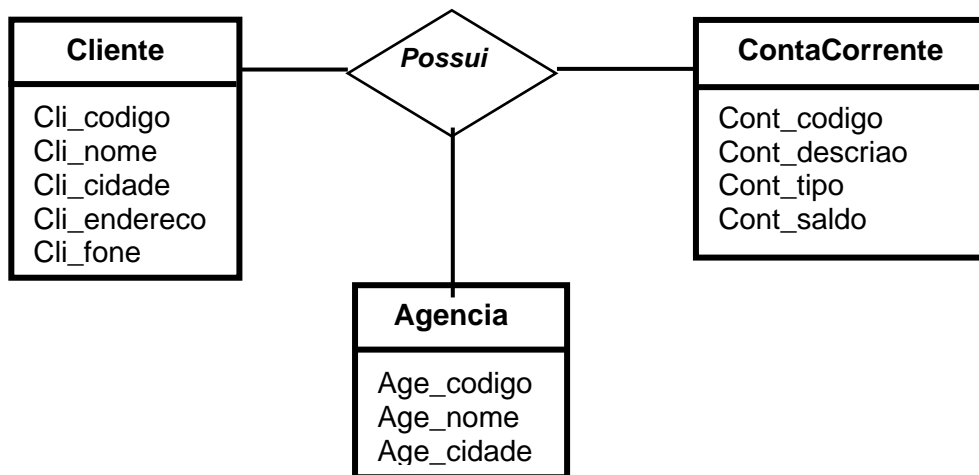


Figura 2.12 - Relacionamento Ternário: “Cliente” - “Agencia” - “Conta”

Fonte: “Acessando Bancos de Dados com Ferramentas RAD: Aplicações em Delphi”
Rio de Janeiro, Brasport, 2008 (do autor).

2.7.4 - Cardinalidades

A cardinalidade é a quantificação do relacionamento entre duas entidades; ela dá o número de ocorrências de determinada entidade em outra entidade com ela relacionada. Quanto ao número de associações que podem ocorrer entre duas entidades, na prática podemos considerar apenas três (apesar de a literatura prever cinco):

- 1:1 *Um-para-um*
- 1:N *Um-para-muitos*
- N:M *Muitos-para-muitos*
- **Cardinalidade 1:1** - Ocorre quando uma entidade **A** “aparece” apenas uma vez na entidade **B**, e a entidade **B** também “aparece” apenas uma vez na entidade **A**. É o caso por exemplo, do relacionamento entre as entidades “Esposa” e “Marido”, cuja semântica pode ser descrita assim: *“uma esposa possui apenas um marido e um marido possui apenas uma esposa”*.
- **Cardinalidade 1:N** - Ocorre quando uma entidade **A** pode aparecer muitas vezes na entidade **B**, mas a entidade **B** só pode aparecer apenas uma vez na entidade **A**. É o que acontece entre as entidades “Cliente” e “ContaReceb”, já explicado anteriormente
- **Cardinalidade N:M** - Ocorre quando duas entidades se relacionam de tal forma que a entidade **A** pode aparecer muitas vezes na entidade **B**, e a entidade **B** também pode aparecer muitas vezes na entidade **A**. Esta situação foi descrita na **Figura 2.8b** no relacionamento entre as entidades “Medico” e “Paciente”, onde um médico pode tratar vários pacientes e um paciente também pode ser tratado por vários médicos.

Apesar de poder existir num mesmo modelo de dados os três tipos de cardinalidades descritos acima, na prática a cardinalidade 1:N é a mais considerada, evitando assim, complicações desnecessárias.

Na cardinalidade **1:1** uma das entidades pode ser convertida em atributo da outra; por

exemplo, no caso do relacionamento entre as entidades “Esposa” e “Marido”; um dos dois pode ser convertido no atributo “cônjuge”. Entretanto, em aplicações em que as informações sobre um dos cônjuges são realmente relevantes, a cardinalidade **1:1** pode ser considerada; e neste caso deve-se manter uma tabela com os dados do marido e outra com os dados da esposa para que os dados de um dos cônjuges possam ser “puxados” através de um componente *dataware* da ferramenta de desenvolvimento visual empregada, como o Delphi ou VB.

A cardinalidade **N:M** é muito difícil de ser tratada em nível de programação; por isto, o que se faz é criar uma nova entidade (entidade associativa) que se relacionará com as outras com cardinalidade **N:1**, tal como foi mostrado no esquema da **figura 2.8b**.

Notações de cardinalidades

Existem várias notações para indicar a cardinalidade nos relacionamentos entre as entidades: notação relacional, IDEF1X e “Pés de Galinha”. Neste trabalho será usada essa última, por ser a mais simples e menos “carregada”. Essa notação emprega conectores (os pés de galinha) nas extremidades do segmento de reta que une as entidades que se relacionam. Veja o esquema da **figura 2.13**.

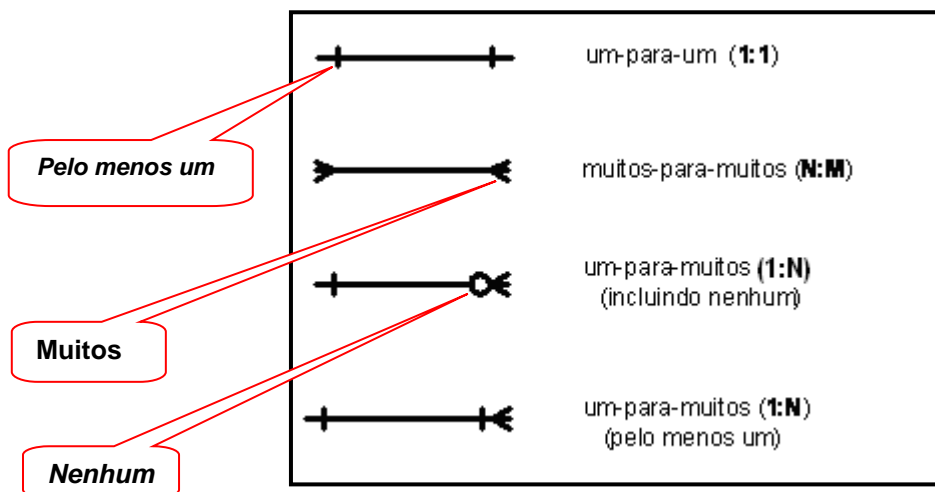


Figura 2.13 - Conectores usados nos relacionamentos

Fonte: “Acessando Bancos de Dados com Ferramentas RAD: Aplicações Em Delphi”
Rio de Janeiro, Ed. Brasport, 2008 (do autor).

A **Figura 2.13** mostra os três tipos de cardinalidades comentados anteriormente: **1:1**, **1:N** e **N:M**. O *tracinho* vertical sobre a linha indica **PELO MENOS UM**, o “pé de galinha” indica **MUITOS** e a “bolinha” sobre a linha quer dizer **NENHUM**. E como já foi explicado, a cardinalidade empregada na prática é a “*um-para-muitos*” (1:N), aí incluídas as duas alternativas: “um-para-muitos” (com pelo menos um) e “um para-muitos” (incluindo nenhum).

2.8 - Elementos de um Banco de Dados

Os elementos básicos de um banco de dados podem ser indicados como três:

- Tabelas
- Chaves
- Índices

2.8.1 - Tabelas

As tabelas são os principais elementos de um banco de dados, pois são elas que armazenam efetivamente os dados, embora ainda existam no mercado GA's que controlam tabelas separadas, que não se relacionam diretamente entre si como são os casos do formato xBase (dBase, xHarbour e Fox Pro), Paradox, PostgreSQL e outros. Nesses casos, ao invés de se ter um arquivo único, tem-se arquivos separados onde são armazenados os dados. Deste modo, a responsabilidade cruzar os dados e processá-los adequadamente para se obter informações gerenciais fica por conta do aplicativo e não do sistema gerenciador, tornando o processo de extração das informações um pouco mais trabalhoso. De qualquer forma, uma tabela pode ser considerada “fisicamente” como sendo um conjunto de linhas e colunas, formando uma matriz bidimensional. Essas linhas são os registros que representam entidades particulares, e as colunas os campos armazenam, efetivamente, os dados de uma mesma natureza (de um mesmo tipo de dado) para cada registro. Em resumo, um conjunto de campos forma um registro e um conjunto de registros formam uma tabela. A **figura 2.14** mostra um exemplo de tabela obtida de um banco de dados no formato Access, que armazena os dados das vendas aos clientes.

Venda						
Venda_num	Venda_data	Venda_valo	Cli_Codigo	TipoVenda_Codi	Vendo_mat	
1	22/10/2019	R\$546,34	2	4	1	1
2	22/10/2019	R\$954,47	1	4	1	1
3	22/10/2019	R\$1.200,00	2	4	3	3
4	23/10/2010	R\$854,23	2	3	3	3
5	23/10/2010	R\$1.350,00	1	4	3	3
6	25/10/2010	R\$524,16	4	2	4	4
7	19/11/2019	R\$150,00	13	1	2	2
8	20/12/2019	R\$1.234,45	1	3	2	2
9	20/11/2019	R\$236,48	13	3	1	1

Figura 2.14 - Tabela “Vendas” de um banco Access

2.8.2 - Chaves

São campos com características que os tornam especiais para pesquisas no banco; por exemplo, na tabela “Medico” mostrada na **figura 2.15**, o campo “Med_crm” é uma chave, assim como o campo “Med_cpf”^[4]. De um modo geral podemos considerar três tipos de chaves: chave primária, chave secundária e chave estrangeira.

Medico
#Med_crm
Med_nome
Med_fone
Med_cpf
*Esp_codigo

Figura 2.15 - Tabela “Medico”

⁴ A rigor esses dois campos (Med_crm e Med_cpf) são ambos, chaves candidatas, pois qualquer um deles poderia definir univocamente um **medico**; qualquer um dos dois pode ser definido como chave primária da tabela.

- **Chave Primária (PK)**

A chave primária (Primary Key) é o atributo mais importante de uma tabela, pois é através dele que um registro pode ser determinado univocamente. Duas restrições devem ser consideradas numa PK:

- ✓ Não pode conter valores nulos;
- ✓ Não pode ter valores duplicados.

Assim sendo, o atributo (campo) “Med_crm” na **figura 2.15** pode ser a chave primária da tabela “Medico”, pois todo médico deve ter uma inscrição no Conselho Regional de Medicina de sua região, e esta inscrição deve ser única para cada um deles.

- **Chave secundária (SK)**

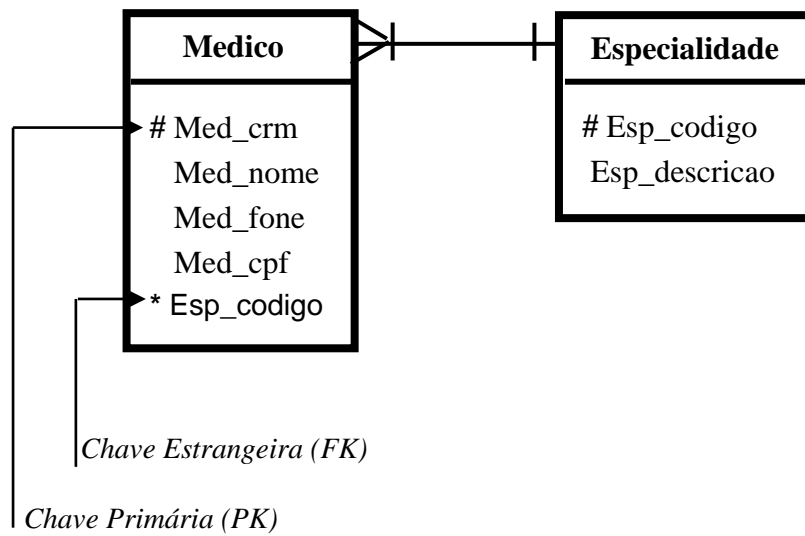
Também chamada de “Índice Secundário” no ambiente de processamento de dados, é um campo através do qual a tabela pode ser classificada, dando um melhor arranjo na exibição dos dados. Por exemplo, um relatório ordenado pelos nomes dos clientes. Mas, ao contrário da chave primária, podem existir várias chaves secundárias e, além disso, podem aparecer valores duplicados. Por exemplo, um cliente de nome “Luiz Carlos da Silva” (nome bastante comum) pode aparecer várias vezes numa mesma tabela, e mesmo assim o campo “Cli_nome” poderia ser definido como uma chave secundária. Por outro lado, deve ser assinalado que para elas não existe uma notação específica.

- **Chave estrangeira (FK)**

Chaves estrangeiras (Foreign Key) são campos da tabela através dos quais ela se relaciona com outra(s) tabela(s) de um banco de dados relacional. Por exemplo, no caso do banco mostrado na **Figura 2.7** a tabela “Vendedor” se relaciona com a tabela “Venda”; um vendedor pode aparecer várias vezes na tabela “Venda”. Este tipo de chave é fundamental na implementação de aplicações que acessam bancos de dados relacionais, permitindo mapear outros campos de interesse. A chave estrangeira estabelece uma associação entre duas tabelas, de modo a identificar a entidade fraca no contexto. Por exemplo, na **Figura 2.16** o relacionamento entre “Medico” e “Especialidade” temos a seguinte situação:

“Especialidade” ==> Entidade forte (possui a chave primária: Esp_codigo)
 “Medico” ==> Entidade fraca (possui a chave estrangeira: Esp_codigo)

A FK está na entidade “Medico”; mas observe que ela é chave primária (PK) na entidade “Especialidade”. Uma chave estrangeira sempre é chave primaria em outra entidade.



Nota: Observe que foi utilizado o mesmo nome de atributo para indicar PK e FK nos relacionamentos entre duas entidades. Apesar de não ser obrigatória, essa convenção facilita enormemente a identificação do campo através do qual ocorre o relacionamento; afinal é o mesmo campo (com as mesmas propriedades), apenas em entidades diferentes; isto é, um atributo *estranho* na tabela. Observe isto na **figura 2.14** com relação ao código do cliente, código do tipo de venda e matrícula do vendedor.

Capítulo 3 - Normalização

3.1 - Conceitos Básicos

Devido ao aumento da complexidade dos bancos de dados, foi preciso estabelecer regras para o armazenamento correto dos dados. A Normalização é uma técnica que tem como objetivo corrigir as disposições dos dados num banco, eliminando as chamadas “anomalias” de armazenamento. É um dos conceitos mais importantes na modelagem de dados, e foi introduzido por E. F Codd em 1972, que inicialmente criou as três primeiras regras (formas) de normalização que foram chamadas de primeira forma normal (1FN), segunda forma normal (2FN) e terceira forma normal (3FN). Outras formas normais surgiram com a crescente complexidade dos bancos de dados corporativos. A normalização de dados é uma série de passos que deve ser executada após o projeto de um banco de dados, para permitir um armazenamento consistente, mais eficiente e sem inconsistências no acesso aos dados em bancos relacionais.

Consideremos o MER da **figura 2.7**; se por acaso o projetista do banco de dados colocasse na entidade “Venda” o atributo “Vendo_nome” do vendedor, estaria cometendo um grave erro (uma anomalia) que feriria a chamada 3ª Forma Normal (3FN). E se ocorresse alguma alteração no nome do vendedor (por exemplo, se o vendedor se chamasse “Martírio dos Prazeres” e resolvesse mudar seu nome??!). Essa alteração teria que ser feita nas duas entidades, pois num relatório de vendas desse vendedor certamente haveria perda de informações!

Portanto, quando um conjunto de dados é usado para ser representado por um banco devemos decidir sobre a forma lógica mais adequada para conter a estrutura desses dados. Essa tarefa pode ser extremamente complexa, mas a Normalização ajuda muito a minorar as dificuldades. Atualmente existem muitas regras (formas normais), muito embora as três primeiras ainda podem resolver 85% dos casos de correção dos dados armazenados^[5]:

- 1ª Forma Normal (1FN)
- 2ª Forma Normal (2FN)
- 3ª Forma Normal (3FN)
- 4ª Forma Normal (4FN)
- 5ª Forma Normal (5FN)

Não importa qual seja a forma norma, a regra fundamental da Normalização é a seguinte:

“Todas as relações estão numa forma normal se estiverem na anterior, acrescida das restrições do seu nível”. Isto quer dizer: “Para que uma relação esteja numa forma N, deverá antes, estar na forma N-1”

3.2 - Primeira Forma Normal (1FN)

“Uma relação está na 1ª Forma Normal se, e somente se, todos os domínios básicos possuírem apenas valores atômicos”.

Do ponto de vista prático, esta definição quer dizer o seguinte:

“Uma relação está na 1ª Forma Normal se e somente se, estiver integrada por tabelas de modo que em cada uma dessas tabelas as colunas contêm valores simples e todas as linhas diferentes”.

⁵ Aqui neste trabalho citaremos também a 4ª e a 5ª Formas Normais; entretanto, como cita o texto, na grande maioria das vezes a normalização até a 3ª Forma Normal atente bem à maioria das exigências da modelagem de dados.

Ordem de Compra nº 721		
Número do Fornecedor: F123 Nome do Fornecedor: InfoTecno Data da Compra: 22/05/09		
Peça	Preço	Quantidade
PX4-II	450,00	200
MX3-1G	180,00	100
CDR-100	110,00	50

Figura 3.1 - Exemplo 3.1: Ordem de Compra

A representação não normalizada da tabela “OrdemCompra” pode ser a da **tabela 3.1**.

- **OC_num** Número da ordem de compra
- **For_cod** Código do fornecedor da peça
- **For_nome** Nome do fornecedor da peça
- **Loja_cod** Código da loja que emitiu a ordem de compra
- **Loja_nome** Nome da loja que emitiu a ordem de compra
- **OC_data** Data de emissão da ordem de compra
- **Peça_cod** Código da peça
- **Peça_preço** Preço unitário da peça
- **Quant** Quantidade de peças

For_cod	For_nome	Loja_cod	Loja_nome	OC_data	Peça_cod	Peça_preço	Quant
F120	TecnoDrive	01	Centro	21/05/09	DRV-100	110,00	50
F123	InfoTecno	01	Centro	22/05/09	PX4-II	450,00	200
					MX3-IG	180,00	100
					CDR-100	210,00	50
F121	Shaolim Inc.	04	Liberdade	25/05/09	Mouse-OpX	85,00	30
F122	Import Lisis	02	Avenida	28/05/09	DRV-100	110,00	40
F123	InfoTecno	03	Higienópolis	04/06/09	MX3-IG	180,00	60
					CDR-100	210,00	100

Tabela 3.1 - Representação “física” de “OrdemCompra”

A notação relacional (tabular) de “OrdemCompra” é a seguinte:

OrdCompra(OC_num,For_cod,For_nome,OC_data,
Loja_cod,Loja_nome,{Peça_cod,Peça_preço, Quant})

O grupo de atributos indicado na “relação”^[6] entre chaves **{Peça_cod, Peça_preço, Quant}** é um grupo de atributos multivalorados (grupo repetido), que impede da relação estar na 1FN. Observem que na **tabela 3.1** os campos Peça_cod, Peça_preço e Quant determinam “tabelas aninhadas” dentro da tabela maior “OrdemCompra”, para a ordem de compra **721**. A notação relacional (tabular) mostra esses campos **{Peça_cod, Peça_preço, Quant}**, dentro de chaves; é um indicativo de que esses campos formam uma segunda tabela dentro da tabela original. Isto quer dizer que a tabela “OrdemCompra” é uma tabela não normalizada. O mesmo acontece com a ordem de compra 724 (veja na **Tabela 3.2**); os três campos são **multivalorados**, o que é proibido pela 1ª Forma Normal.

⁶ Na realidade, em situações como essa não se pode falar, formalmente, em relação, uma vez que viola a regra de definição desse objeto.

Como eliminar o grupo de atributos repetidos?

Solução: Aplicar a 1ª Forma Normal para remover o grupo repetido (atributos multivalorados), fazendo com que esses atributos integrem uma nova tabela, cuja chave primária seja o atributo determinante desse grupo (Peça_cod), concatenado com a chave primária original (OC_num).

Representação em Notação Relacional.

Em termos de notação relacional, a solução é dividir a relação (representada fisicamente pela tabela) em duas: “OrdemCompra” e “OrdemPeça”.

OrdemCompra (OC_num, For_cod, For_nome, OC_data, Loja_cod, Loja_nome)

OC_num	For_cod	For_nome	OC_data	Loja_cod	Loja_nome
720	F120	Tecno Drive	21/05/09	01	Centro
721	F123	InfoTecno	22/05/09	01	Centro
721	F123	InfoTecno	22/05/09	01	Centro
721	F123	InfoTecno	22/05/09	01	Centro
722	F121	Shaolim Inc.	25/05/09	04	Liberdade
723	F122	Import Lisis	28/05/09	02	Avenida
724	F123	InfoTecno	04/06/09	03	Higienópolis
724	F123	InfoTecno	04/06/09	03	Higienópolis

Tabela 3.2 - “OrdemCompra”

OrdemPeça (OC_num, Peça_cod, Peça_preco, Quant)

OC_num	Peça_cod	Peça_preco	Quant
720	DRV-100	110,00	50
721	PX4-II	450,00	200
721	MX3-IG	180,00	100
721	CDR-100	210,00	50
722	Mouse-OpX	85,00	30
723	DRV-100	110,00	40
724	MX3-IG	180,00	60
724	CDR-100	210,00	100

Tabela 3.3 - “OrdemPeça”

A questão que se coloca agora é a seguinte:

“O problema já está totalmente resolvido?” A resposta é **NÃO**; ainda existem anomalias que devem ser eliminadas, como as citadas a seguir:

- **Anomalia de Inserção:** Na tabela “OrdemPeça” o preço de um produto só depende de parte da chave (dependência parcial): depende só do atributo Peça_cod e não da chave primária integralmente.(OC_num+Peça_cod), gerando redundâncias. O mesmo acontece em relação ao fornecedor na tabela “OrdemCompra”: o nome de um fornecedor não depende do número da ordem de compra.
- **Anomalia de Exclusão:** Na tabela “OrdemCompra” se removermos por exemplo, a OC 722, perderemos informações sobre a quantidade comprada e os dados do fornecedor de código **F121**.

- **Anomalia de Atualização:** Ainda na tabela “OrdemCompra” ocorrem redundâncias de informações sobre os nomes dos fornecedores (nomes repetidos).

A questão é esta: qual seria a solução, agora?!

Solução: Resposta: Aplicar um método para colocar a estrutura na 2ª Forma normal.

3.3 - Segunda Forma Normal (2FN)

“Uma relação está na 2ª Forma Normal se, e somente se, estiver na 1ª Forma Normal e se todos os atributos não chaves forem dependentes integralmente da chave primária”

Esta definição implica dizer o seguinte:

“Um modelo de dados está na 2ª Forma Normal se estiver na 1ª Forma Normal e, para cada tabela componente os valores em cada linha para qualquer coluna não chave depender integralmente da chave primária completa, e não de parte dela ⁽⁷⁾”. Esta definição implica afirmar que uma tabela para estar na 2FN não pode conter dependências parciais dos campos com relação à chave primária.

Analisemos o caso da tabela “OrdemPeça” (esquematizada na **tabela 3.3**) Essa tabela não está na 2FN, pois existe o campo **Peça_preço** que depende só de parte da chave primária (só do código de peça). O mesmo ocorre com relação à tabela “OrdemCompra”, onde o nome do fornecedor só depende do seu código. Então devemos eliminar essas dependências parciais, criando outra tabela adicional. Assim, finalmente, o resultado do nosso banco da 2FN é descrito fisicamente pelas tabelas **3.4**, **3.5**, **3.6** e **3.7**.

OC_num	Peça_cod	Quant
720	DRV-100	50
721	PX4-II	200
721	MX3-1G	100
721	CDR-100	50
722	Mouse-OpX	30
723	DRV-100	40
724	MX3-IG	60
724	CDR-100	100

Tabela 3.4

For_cod	For_nome	Loja_cod	Loja_nome	OC_data	Peça_cod	Peça_preço	Quant
F120	TecnoDrive	01	Centro	21/05/09	DRV-100	110,00	50
F123	InfoTecno	01	Centro	22/05/09	PX4-II	450,00	200
					MX3-IG	180,00	100
					CDR-100	210,00	50
F121	Shaolim Inc.	04	Liberdade	25/05/09	Mouse-OpX	85,00	30
F122	Import Lisis	02	Avenida	28/05/09	DRV-100	110,00	40
F123	InfoTecno	03	Higienópolis	04/06/09	MX3-IG	180,00	60
					CDR-100	210,00	100

Tabela 3.5

⁷ Um caso particular é quando a tabela estiver na 1FN e possuir chave primária simples (apenas um atributo). Neste caso automaticamente ela também estará na 2FN, pois não há possibilidade de haver dependência funcional parcial.

OC_num	For_cod	For_nome	OC_data	Loja_cod	Loja_nome
720	F120	Tecno Drive	21/05/09	01	Centro
721	F123	InfoTecno	22/05/09	01	Centro
721	F123	InfoTecno	22/05/09	01	Centro
721	F123	InfoTecno	22/05/09	01	Centro
722	F121	Shaolim Inc.	25/05/09	04	Liberdade
723	F122	Import Lisis	28/05/09	02	Avenida
724	F123	InfoTecno	04/06/09	03	Higienópolis
724	F123	InfoTecno	04/06/09	03	Higienópolis

Tabela 3.6

OC_num	Peça_cod	Peça_preco	Quant
720	DRV-100	110,00	50
721	PX4-II	450,00	200
721	MX3-IG	180,00	100
721	CDR-100	210,00	50
722	Mouse-OpX	85,00	30
723	DRV-100	110,00	40
724	MX3-IG	180,00	60
724	CDR-100	210,00	100

Tabela 3.7

Com relação às **tabela 3.5** e **3.6** ainda termos a seguinte situação: elas estão na 2FN, portanto estão, também, na 1FN; mas notamos que existe uma anomalia estranha: O nome da loja que emitiu a ordem de compra (**Loja_nome**) depende funcionalmente do código da loja (**Loja_cod**) e que não é chave primária na relação. Isto é conhecido como **Dependência Transitiva**.

“Existe dependência transitiva quando um atributo depende de outro atributo chave que não é chave primária nessa tabela onde eles estão”.

3.4 - Terceira Forma Normal (3FN)

“Uma relação está na 3ª Forma Normal se e somente se, estiver na 2ª Forma Normal e todos os atributos não chaves forem dependentes não transitivos da chave primária”. Em outras palavras:

“Um modelo está na 3FN se está na 2FN e se para cada tabela, considerando apenas as colunas não chaves, estas forem mutuamente independentes; isto é, se não existirem dependências transitivas”.

Para eliminar uma dependência transitiva, criamos uma nova tabela contendo os atributos dependentes e tendo como chave primária o atributo que define funcionalmente esses atributos dependentes. Então, para resolver o problema na tabela “OrdemCompra”, criaremos uma nova tabela denominada “Fornecedor” que terá como atributos apenas o código e o nome do fornecedor. Em notação relacional, obteremos o seguinte:

OrdemCompra (OC_num, For_cod, OC_data)
Loja (Loja_cod, Loja_nome)

Finalmente, a estrutura do nosso banco de dados estará normalizada (até a 3FN)

OrdemCompra(OC_num, For_cod, OC_data)
Loja(Loja_cod, Loja_nome)
Fornecedor(For_cod, For_nome)
OrdemPeça(Oc_num, Peça_cod, Quant)
Peça(Peça_cod, Peça_preço)

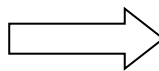
3.5 - Quarta Forma Normal (4FN)

A Quarta Forma Normal deve ser aplicada em bancos de dados que possuam tabelas com relacionamentos ternários (como no caso clássico de **Cliente-Agência-Conta**) e que possuam dependências funcionais multivaloradas. Além disso, é necessário que essas tabelas possuam chave primária tripla (composta de três atributos) e que não possuam atributos não chaves. Nesses casos ocorre também o aparecimento de atributos multivalorados, os quais passam a fazer parte da chave primária; isto determina a necessidade de aplicar a 4FN.

Considere a seguinte situação: O paciente “Luiz” possui três diferentes planos de saúde (Plano A, Plano B e Plano C), e necessita fazer três diferentes tipos de exames (sangue, urina e colesterol). As **tabelas 3.8a e 3.8b** esclarecem a situação inicial.

Paciente	Plano	Exame
Luiz	Plano A	Sangue
Luiz	Plano A	Urina
Luiz	Plano A	Colesterol
Luiz	Plano B	Sangue
Luiz	Plano B	Urina
Luiz	Plano B	Colesterol
Luiz	Plano C	Sangue
Luiz	Plano C	Urina
Luiz	Plano C	Colesterol

Tabela 3.8a - Tabela “Paciente1” original



Paciente	Plano	Exame
Luiz	Plano A	Sangue
		Urina
		Colesterol
	Plano B	Sangue
		Urina
		Colesterol
	Pano C	Sangue
		Urina
		Colesterol

Tabela 3.8b - Tabela “Paciente1” estilizada

A tabela “Paciente1” está na 3FN, pois não existem dependências multivaloradas, mas ocorrem anomalias devido a valores multivalorados: a um único paciente estão ligados mais de um plano e mais de um tipo de exame para cada plano. Observe isto na **Tabela 3.8b**, onde é mostrado claramente o aninhamento das tabelas. Além disso, a chave primária da tabela é composta por três atributos, (Paciente-Plano-Exame). Nesses casos, apesar de estar normalizada até a 3FN, há necessidade de se passar à 4FN.

Para passar à 4FN neste exemplo devem ser projetados subconjuntos de dados que focalizem as ocorrências multivaloradas, individualmente. Essas ocorrências (fatos reais) são duas:

- Plano de Saúde
- Tipo de Exame

Então, o problema pode ser resolvido com duas tabelas: “Plano de Saude” e “Tipo de Exame”, ambas contento os planos e o mesmo paciente, e ambas com chave primária dupla, como mostram as **tabelas 3.9 e 3.10**, respectivamente.

Plano	Paciente
Plano A	Luiz
Plano B	Luiz
Plano C	Luiz

Tabela 3.9 - Tabela “Plano de Saúde”

Exame	Paciente
Sangue	Luiz
Urina	Luiz
Colesterol	Luiz

Tabela 3.10 - Tabela “Tipo de Exame”

3.6 - Quinta Forma Normal (5FN)

A Quinta Forma Normal, tal como a 4FN, só é aplicável em tabelas que se relacionam de modo ternário e sem atributos não-chave. Pela regra: “*uma tabela está na 5FN se, e somente se, estiver na 4FN e se o relacionamento triplo puder ser decomposto em três tabelas de relacionamento binário, com chave primária dupla*”. Esse desmembramento deve ser tal que, se juntarmos novamente as três tabelas numa só não pode haver geração de dados incorretos.

De uma forma mais geral, a teoria diz que um registro está na sua 5FN quando o seu conteúdo não puder ser reconstruído (por junção) a partir de outros registros menores, extraídos desse registro principal. Assim, se ao particionar um registro e sua junção posterior não conseguir recuperar as informações contidas no registro original, então este registro está na 5FN. Assim, modo, só haverá necessidade de se passar à 5FN se o banco estiver totalmente na 4FN e existir alguma tabela que não possa ser desmembrada em duas relações sem haver junção delas.

Apesar de existir teoricamente, na prática, porém, a aplicação da 5FN é raramente necessária, uma vez que na modelagem de dados - na maioria das vezes – é possível criar o banco com apenas relacionamentos binários.

Conforme foi dito, a 5FN diz respeito à dependência de junção; e para explicar isso vamos considerar a seguinte situação: um fornecedor F_i fornece um certo componente C_i e um projeto P_i encomenda esse componente C_i e o mesmo fornecedor F_i fornece o projeto P_i . Então, nesse caso esse fornecedor F_i fornece o componente C_i ao projeto P_i . Considerando essa situação, observe o esquema da **tabela 3.11**.

Fornecedor	Componente	Projeto
F1	C2	P1
F1	C3	P2
F2	C1	P1
F3	C2	P2
F2	C1	P1
F3	C1	P2

Tabela 3.11 - Relação “Fornecimento”

Considerando a situação descrita pela **Tabela 3.11**, então as duas últimas tuplas devem existir para qualquer situação da relação “Fornecimento”. Isto nos leva à seguinte conclusão: existe uma dependência de junção do tipo $J(R_1, R_2, R_3)$, onde R_1 , R_2 e R_3 são relações definidas assim:

R1(Fornecedor, Componente)

R1(Fornecedor, Projeto)

R1(Componente, Projeto)

O esquema da **Figura 3.2** ilustra a situação deste exemplo.

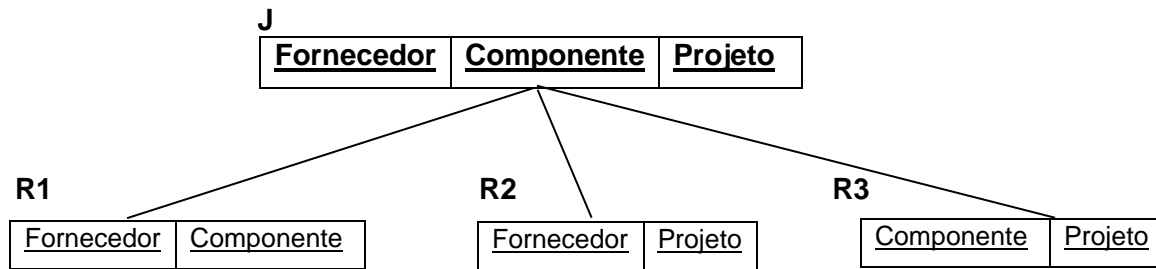


Figura 3.2 - Esquema de passagem à 5FN com dependência de junção

Conclusão: A junção inicial **J** com chave tripla deu origem a três relações (**R1**, **R2** e **R3**), cada uma com chave dupla.

Desse modo, uma outra definição para a Quinta Forma Normal seria a seguinte:

“Um esquema relacional R está na quinta forma normal relativamente a um conjunto F de dependências funcionais multivaloradas e de junção, se estiver na quarta forma normal e se para cada dependência de junção não-trivial $J(R_1, R_2, \dots, R_n)$ em F , cada R_i ($i=1,n$) é super-chave”.

3.7 - Algoritmos práticos para aplicar as três primeiras formas normais

Conforme foi dito no **item 3.1**, embora existam mais de três formas normais, na prática são as três primeiras as mais empregadas para normalizar banco de dados relacionais. Um algoritmo simples para aplicar essas três regras de normalização pode ser resumido nos seguintes passos:

3.7.1 - Para passar à 1FN

Crie uma tabela na 1FN (referente à tabela não normalizada), porém sem as tabelas aninhadas. A chave primária dessa tabela resultante é a mesma da tabela não normalizada; e para cada tabela aninhada crie uma tabela na 1FN composta dos seguintes campos:

- ✓ A chave primária de cada uma dessas tabelas na qual está aninhada;
- ✓ Os campos da própria tabela aninhada.
- ✓ Defina as chaves primárias das tabelas (na 1FN) que correspondem às tabelas aninhadas.

Atenção: A chave primária de uma tabela na 1FN (que era aninhada) não é necessariamente igual à concatenação das chaves primárias da tabela não aninhada + a da tabela aninhada. Isto vai depender da situação que se apresentar na estrutura original.

❖ Exemplo 3.2 - Controle de Concurso Público

Este exemplo ilustra o que foi dito para passar à 1FN, partindo de um arranjo normalizado (na 0FN - com atributos multivalorados).

Considere uma estrutura que representa um arquivo de dados sobre um concurso realizado em uma instituição pública, sujeito às seguintes restrições:

- ✓ Cada especialidade tem um registro contendo o código (Esp_cod), nome da especialidade (Esp_nome) e as vagas oferecidas nessa especialidade (Esp_vagas);

- ✓ Para cada especialidade existe uma listagem dos candidatos recentemente aprovados;
- ✓ Cada candidato tem um registro com sua matrícula (Cand_mat), seu nome (Cand_nome) e o número de pontos obtidos no concurso (Cand_pontos);
- ✓ Cada candidato só pode se inscrever em uma única especialidade.

Arquivo (arranjo) original na 0FN:

Concurso (Esp_cod, Esp_nome, Esp_vagas, {Cand_mat, Cand_nome, Cand_pontos})

Tabela aninhada

Passando à 1FN, teremos:

Especialidade (Esp_cod, Esp_nome, Esp_vagas)

EspCand (Esp_cod, Cand_mat, Cand_nome, Cand_ponto)^[8]

3.7.2 - Para passar à 2FN

Passar à 2FN uma tabela que está normalizada somente na 1FN, requer a eliminação das dependências funcionais parciais. Desse modo, para passar à 2FN uma tabela deve, necessariamente (além de estar normalizada na 1FN), possuir chave primária composta. Para isso devemos fazer o seguinte:

1. Retirar da chave o campo que determina funcionalmente outros campos;
2. Criar nova tabela em que o campo retirado, será chave primária nesta nova tabela.

❖ Exemplo 3.3 - Controle de Nota Fiscal (na 2FN)

Considere o esquema relacional abaixo e a tabela “ItemNF” (itens da nota fiscal) visto no esquema na **tabela 3.12**.

ItemNF(NF_num, Prod_cod, Prod_desc, Prod_preço, Cli_cod, Cli_nome, Quant, Total)

NF_num	Prod_cod	Prod_desc	Prod_preço	Cli_cod	Cli_nome	Quant	Total
NF1234	Ele455	Cafeteira VX	75,00	1248	José Carlos	2	150,00
NF1234	Ele255	Micro-Ondas 10L	456,00	1249	Luiz Carlos	1	456,00
NF1234	Ele323	Refrigerador SP	950,00	1235	Luiza Parisi	1	950,00
NF1235	Ele275	TV 52" LCD	5.800,00	1422	Ana Lúcia	1	5.800,00
NF1236	Mov255	Estante LX	482,00	1357	Carlos José	1	482,00
NF1237	Ele255	Micro-Ondas 10L	456,00	1143	Nelson Nunes	1	456,00

Tabela 3.12

Pela **tabela 3.12** podemos ver que para identificar perfeitamente uma linha (*tupla*) da tabela “ItemNF” são necessários dois campos (atributos): número da nota fiscal (NF_num) e o código do produto (Prod_cod); isto é, a chave dessa tabela é “NF_num+Prod_cod”. Pois, por exemplo, o item vendido do produto **Ele255** aparece duas vezes; então, para identificarmos perfeitamente o item vendido precisamos do número da nota fiscal. Por outro lado, pode ser notado que a tabela, tal como se apresenta, não está na 2FN, pois o

⁸ Observem que a chave primária da tabela **EspCand** é simples, já que cada candidato só pode se inscrever (e ser aprovado) num único curso. Portanto, não houve necessidade de concatenar: **Esp_cod+Cand_mat**.

preço do produto (Prod_preço) e a sua descrição (Prod_desc) não dependem integralmente da chave primária; isto é, dependem apenas parcialmente (somente do seu código e não da nota fiscal). Aí estão caracterizadas dependências parciais, que devem ser eliminadas.

Passando à 2FN obteremos o esquema seguinte relacional normalizado.

NotaFiscal (NF_num, NF_data, Prod_cod, Cli_cod, Cli_nome)
Produto (Prod_cod, Prod_desc, Prod_preço)
ItemNF (NF_num, Prod_cod, Quant, Total)

3.7.3 - Para passar à 3FN

Conforme já mencionando, para passarmos uma tabela para a 3FN (estando já na 2FN), devemos eliminar as chamadas dependências transitivas. Retomando o exemplo “Controle de Nota Fiscal” (**Exemplo 3.3**), notamos que o esquema relacional obtido, ainda possui a tabela (NotaFiscal) que, apesar de estar normalizada na 2FN, não está na 3FN, porque possui determinante não-chave^[9], o que implica dependências transitivas. Para passar à 3FN, devemos proceder do seguinte modo:

1. Para cada determinante não-chave, remover da relação os atributos que dependem desse determinante;
2. Cria uma nova relação (tabela) contendo os atributos da relação original que dependem desse determinante;
3. Tornar o determinante a chave primária da nova relação.

❖ Exemplo 3.4 - Controle de Nota Fiscal (na 3FN)

Considerando o modelo obtido no **Exemplo 3.3**, vemos que a única relação que não está na 3FN é a relação NotaFiscal, porque possui um determinante não-chave (Cli_cod). Esse atributo determina funcionalmente o atributo Cli_nome (nome do cliente). Desse modo, conforme mencionado acima, devemos remover esse atributo da relação para integrar uma nova relação, cuja chave primária será esse determinante não-chave (Cli_cod). Desta maneira, finalmente obteremos nosso modelo de dados normalizado até a 3F, com a criação da nova relação “Cliente”.

NotaFiscal (NF_num, NF_data, Prod_cod, Cli_cod)
Produto (Prod_cod, Prod_desc, Prod_preço)
ItemNF (NF_num, Prod_cod, Quant, Total)
Cliente (Cli_cod, Cli_nome)

Atributo que dependia funcionalmente de **Cli_cod** na relação “NotaFiscal”

⁹ Determinante não-chave é aquele atributo da relação que não faz parte da chave primária da relação, mas que determina funcionalmente outros atributos dessa relação.

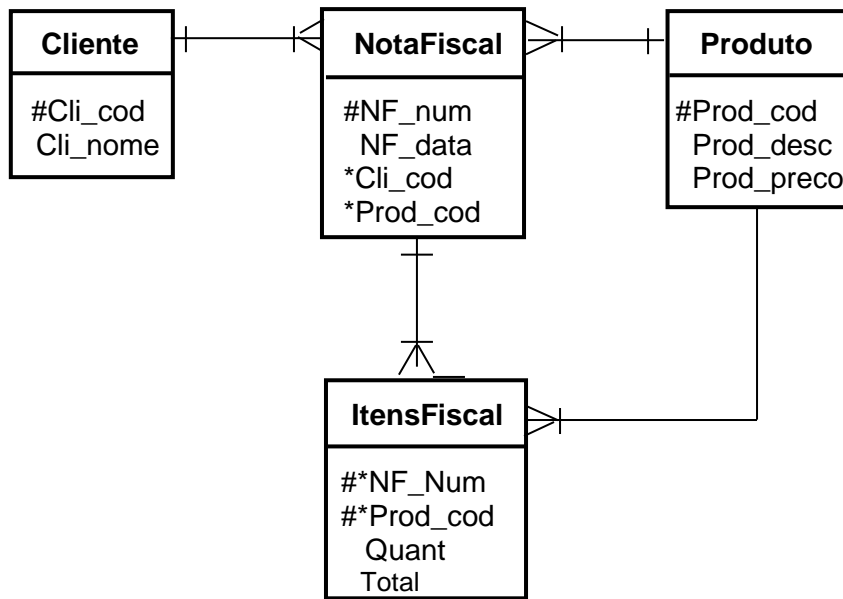


Figura 3.3 - O MER do banco normalizado

Nota: Um exemplo prático em que na maioria das vezes o programador “atropela” a 3FN é com relação aos campos calculados numa tabela. Por exemplo, numa tabela de vendas onde se tem o campo “Quantidade” e o campo “Preço Unitário” do produto, sempre “dá vontade” de criar o campo “Total” como sendo um produto desses dois campos (como se faz normalmente em planilhas eletrônicas). A 3FN “proíbe” isto, mas na prática, e por motivos de melhoria na performance da aplicação, esse campo sempre é criado (pelo menos como campo virtual). Isto pode ser considerado um “pecadinho” de modelagem, mas não um pecado mortal. Afinal, se você é candidato a “santo” e não quer cometer nem mesmo esse “pecadinho”, empregue Campos Calculados; um recurso muito usado nas aplicações em Delphi.

Capítulo 4 - Linguagem de Consulta Estruturada

4.1 - Conceitos Básicos

A linguagem SQL (**Structured Query Language** - Linguagem de Consulta Estruturada) é a mais empregada para fazer consultas em bancos de dados relacionais. Desenvolvida pela IBM e colocada no mercado em 1976, se afirmou de tal maneira no mercado que se tornou padrão para consultas em bancos de dados relacionais. Algumas características dessa linguagem podem ser listadas:

- É baseada no inglês; fácil de escrever, ler e entender;
- Tem como características a economia de tempo, flexibilidade e segurança na manutenção de bancos de dados.
- Existem pequenas diferenças da linguagem em alguns bancos. No VisualFox por exemplo, quando o comando ocupa mais de uma linha devemos colocar ponto e vírgula no final de cada linha, e no final do comando somente um **<Enter>**. No Oracle devemos colocar um ponto e vírgula somente no final do comando; o mesmo vale para o MySQL, Interbase e Firebird.

SQL é uma linguagem de pesquisa (considerada de 4ª Geração) declarativa para bases de dados relacionais, e muitas de suas características originais foram inspiradas no cálculo de *tuplas*, a partir da álgebra de banco de dados. Por outro lado, embora a SQL tenha sido originalmente criada pela IBM, rapidamente surgiram vários "dialetos" desenvolvidos por outros produtores. Essa expansão levou à necessidade de ser criado e adaptado um padrão para a linguagem, tarefa realizada pelo **American National Standards Institute (ANSI)** em 1986 e pela ISO em 1987. Em 1992 ocorreu uma revisão, e a essa versão foi dado o nome de SQL-92; revisto. Outra revisão ocorreu novamente em 1999 e em 2003 para se tornar SQL:1999 (SQL3) e SQL:2003, respectivamente.

O SQL:1999 usa expressões regulares^[10] de emparelhamento, *queries* recursivas e *triggers* ^[11]. Também foi feita uma adição controversa de tipos não-escalados e algumas características de orientação a objetos. O SQL:2003 introduz características relacionadas à linguagem XML, seqüências padronizadas e colunas com valores de auto-generalização (inclusive colunas-identidade). E como dito anteriormente, a SQL, embora padronizada pela ANSI e ISO, possui muitas variações e extensões produzidos pelos diferentes fabricantes de sistemas gerenciadores de bases de dados. Tipicamente, a linguagem pode ser migrada de plataforma para plataforma sem mudanças estruturais principais. Uma vantagem adicional dessa linguagem é permitir interações com o banco de dados; por exemplo, o Oracle e outros incluem Java na base de dados, enquanto o PostgreSQL permite que funções sejam escritas em Perl, TCL e C, entre outras linguagens. Ferramentas RAD, tal como o Visual Basic e o Delphi, também permitem instruções baseadas em comandos da linguagem SQL de dentro as aplicações; os cuidados que devem ser tomados é com relação às pequenas alterações que existem para essas duas ferramentas. Por exemplo, o curinga para o Delphi é o símbolo de percentual (%); já para o VB é o asterisco (*). Mas as diferenças são mínimas, não inviabilizando a migração de uma ferramenta para outra.

¹⁰ Uma **expressão regular**, na Informática, define um padrão a ser usado para procurar ou substituir palavras ou grupos de palavras. É um meio preciso de se fazer buscas de determinadas porções de texto.

¹¹ Triggers serão vistos no Capítulo 5.

4.2 - Subdivisões da SQL: DDL e DML

SQL apresenta uma série de comandos que permitem a definição dos dados, chamada de **DDL** (**Data Definition Language**), composta entre outros, pelos comandos CREATE que é destinado à criação do banco e das tabelas que o compõe, além das relações existentes entre as tabelas. Como exemplos de comandos da classe DDL temos: CREATE, ALTER TABLE e DROP.

Os comandos da classe **DML** (**Data Manipulation Language**) são destinados a consultas, inserções, exclusões e alterações em um ou mais registros de uma ou mais tabelas. Como exemplos de comandos da classe DML podem ser citados os comandos SELECT, INSERT, UPDATE e DELETE. Uma subclasse de comandos DML, a **DCL** (**Data Control Language**), dispõe de comandos de controle como GRANT E REVOKE. A linguagem SQL é baseada em *comandos, cláusulas, predicadas, funções agregadas e operadores*. As **tabelas 4.1, 4.2 e 4.3** mostram alguns desses elementos que compõe uma instrução *sql*. A **tabela 4.4** mostra os principais operadores utilizados nas instruções.

Comando	Classe	Descrição
SELECT	DML	Extraí um conjunto de registros de uma ou mais tabelas.
INSERT	DML	Inserir dados num registro de uma tabela.
UPDATE	DML	Atualiza os dados de uma tabela.
DELETE	DML	Exclui registros de uma tabela.
CREATE	DDL	Cria banco, tabelas, campos e índices.
DROP	DDL	Exclui tabelas e índices.
ALTER	DDL	Altera a estrutura de uma tabela.

Tabela 4.1 - Comandos de uma instrução sql

Cláusula	Descrição
FROM	Especifica as tabelas que serão usadas como fonte de dados.
WHERE	Especifica as condições (critérios) usadas para extrair os registros que comporão o conjunto desejado.
ORDER BY	Classifica (ordena) os registros do conjunto selecionado.
HAVING	Especifica a condição que cada grupo selecionado deve satisfazer.
GROUP BY	Separa os registros extraídos (selecionados) em diferentes grupos.
EXISTS	Verifica se existe tupla.

Tabela 4.2 - Cláusulas de uma instrução sql

Função	Descrição
AVG(n)	Retorna a média dos valores contidos no campo n (ignorando nulos).
COUNT	Retorna a quantidade de registros de uma seleção.
SUM(n)	Retorna a soma dos valores do campo n (ignorando nulos).
MAX	Retorna o maior valor contido num campo.
MIN	Retorna o menor valor contido num campo.
LOWER	Faz com que caracteres maiúsculos apareçam em minúsculos.
UPPER	Faz com que caracteres minúsculos apareçam em maiúsculos.
NVL	Converte valores nulos em zeros
Concat(x,y)	Concatena a <i>string</i> " x " com a <i>string</i> " y ".
Substring(x,y,str)	Extraí um <i>substring</i> da <i>string</i> " str " de " x " a " y ".

Tabela 4.3 - Funções agregadas numa instrução sql

Tipo de Operador	Operador	Descrição
Lógicos	AND	Conjunção
	OR	Disjunção
	NOT	Negação
Comparação	=	Igual a
	<>	Diferente de
	>	Maior que
	<	Menor que
	>=	Maior ou Igual a
	<=	Menor ou Igual a
Outros operadores	BETWEEN	Especifica um intervalo de valores (inclusive)
	LIKE	Especifica um padrão de comparação
	IN	Especifica registros dentro de um banco de dados
	IS NULL	É um valor nulo.

Tabela 4.4 - Operadores empregados numa instrução *sql*

A linguagem SQL tem como grande virtude sua capacidade de gerenciar índices sem a necessidade de controle individualizado de índice corrente, algo muito comum nas linguagens de manipulação de dados do tipo registro a registro (procedurais). Outra característica muito importante disponível em SQL é sua capacidade de construção de visões, que são formas de obter os dados na forma de listagens independente das tabelas e organização lógica dos dados. Uma outra característica também interessante é a capacidade de cancelar uma série de atualizações ou de gravação depois de iniciada uma sequência de atualizações; os comandos COMMIT e ROLLBACK são responsáveis por esta facilidade. Deve ser anotado que a linguagem SQL consegue implementar essas soluções somente pelo fato de estar baseada em banco de dados, que garante por si mesmo a integridade das relações existentes entre tabelas e índices.

4.3 - Principais comandos

4.3.1 - Comando CREATE DATABASE/TABLE

Este comando permite a criação de banco de dados e de tabelas.

CREATE DATABASE <nome_banco >;

Ex: Cria o banco de dados "Escola"

CREATE DATABASE Escola;

CREATE TABLE < nome_tabela >

(campo1 < tipo> [complemento],

campo2 <tipo> [complemento],

...

...

...

campoN <tipo> [complemento],

PRIMARY KEY(campo)

CONSTRAINT FK_chave **FOREIGN KEY**(campoFK) **REFERENCES** <Tabela-mãe(PK)>

);

nome_table indica o nome da tabela a ser criada.
 campo indica o nome do campo a ser criado na tabela.
 tipo indica a definição do tipo de dado do campo
 CONSTRAINT palavra-chave para criar uma chave estrangeira de nome FK_chave

Por exemplo, criar a tabela “Avaliacoes”

```
CREATE TABLE Avaliacoes
(
  Alu_matr      VARCHAR(10) NOT NULL,
  Disc_codigo   VARCHAR(6)  NOT NULL,
  Aval_nota     DECIMAL(3,1),
  Aval_bim1     DECIMAL(3,1),
  Aval_bim2     DECIMAL(3,1),
  Aval_bim3     DECIMAL(3,1),
  Aval_bim4     DECIMAL(3,1),
  PRIMARY KEY (Alu_matr, Disc_Codigo) NOT NULL,
  CONSTRAINT FK_AluMatr FOREIGN KEY (Alu_matr) REFERENCES Alunos,
  CONSTRAINT FK_DiscCodigo FOREIGN KEY (Disc_codigo) REFERENCES Disciplinas
);
```

Nota: No exemplo acima, os atributos **Alu_mat** e **Disc_codigo** são chaves estrangeiras (chaves primárias nas tabelas “Alunos” e “Disciplinas”, respectivamente. Esses dois atributos, juntos, chave primária da tabela “Avaliacoes”.

4.3.2 - Comando Open

Abre (conecta) um banco de dados que foi previamente criado.
 :

OPEN <nome_banco >; (No MySQL é **USE** <nome_banco >)

4.3.3 - Comando DROP

Este comando elimina a definição da tabela, seus dados e referências.

DROP TABLE < nome_tabela >;

4.3.4 - Comando ALTER TABLE

Este comando permite adicionar/alterar/eliminar campos na estrutura de uma tabela.

Sintaxe abreviada:

```
ALTER TABLE < nome_tabela > ADD / DROP / ALTER / RENAME / CHANGE
nome_atributo1 < tipo > [ NOT NULL],
nome_atributoN < tipo > [ NOT NULL]
```

Entretanto, se for tentado eliminar uma chave primária devidamente referenciada em outra tabela como chave estrangeira, ao invés de obter a eliminação do campo, obterá apenas um erro.

Além do comando DROP que poderá eliminar uma tabela e suas relações, também pode ser criada uma relação que tenha os atributos que se deseja, copiando a relação antiga sobre a nova e apagando a relação que originalmente desejávamos eliminar.

Ex: Alterar a estrutura da tabela 'Alunos' para incluir o atributo "Alu_fone"

```
ALTER TABLE Alunos ADD Alu_fone VARCHAR(13);
```

Ex: Excluir o campo Alu_email da tabela 'Alunos'

```
ALTER TABLE Alunos DROP COLUMN Alu_email;
```

4.3.5 - Comando SELECT

Deve ser sempre enfatizado que a linguagem SQL é utilizada tanto pelos profissionais responsáveis pelos dados (ressaltando as figuras do Administrador e do Analista), quanto pelos desenvolvedores de aplicações. Enquanto aqueles estão preocupados com o desempenho e integridade do banco de dados, e utilizam toda a gama de recursos disponíveis na SQL; estes estão preocupados apenas em "transformar dados em informações". Portanto, para os desenvolvedores costuma-se dizer que "*conhecer o comando SELECT*" é o bastante. Nesta apostila será enfatizada a importância de TODOS os comandos do SQL; entretanto sabemos de antemão que os professores de ferramentas como VB e Delphi, sempre ressaltarão a preponderância da instrução "SELECT". Esse comando seleciona (extrai, exibe) um conjunto de registros, oferecendo várias "visões" dos dados armazenados no banco. Podem ser extraídos dados de uma única tabela ou de várias delas que se relacionam. Essas visões particulares são o resultado da aplicação de restrições e junções de tabelas.

```
SELECT    <lista de campos>
          FROM tabela(s) de onde extrai os dados
          WHERE restrições de tuplas individuais
          GROUP BY condições de agrupamento das tuplas
          HAVING restrições dos conjuntos de tuplas
          ORDER BY campo(s)
```

❖ **Exemplo 4.1** - Selecionar todos os alunos da tabela "Alunos"

```
SELECT * FROM Alunos;
```

❖ **Exemplo 4.2** - Selecionar todos os alunos da tabela "Alunos", do sexo feminino

```
SELECT * FROM Alunos WHERE Alu_sexo = 'F';
```

Observe que no segundo exemplo ocorre a restrição: "*alunos do sexo feminino*"; o campo "Alu_sexo" que armazena o sexo do aluno (M ou F) deve ter o valor 'F' (F colocado entre **apóstrofes**, pois este campo é do tipo CHAR). E como foi mencionado anteriormente, o comando SELECT é normalmente composto de cláusulas, predicadas e funções agregadas. A seguir são apresentados vários exemplos de instruções *sql* empregam esses elementos da linguagem.

4.4 - Equi-Junção (Junção por igualdade)

O relacionamento existente entre tabelas é chamado tecnicamente de **equi-junção**, pois os valores de colunas das duas tabelas são equivalentes. A equi-junção é possível apenas quando tivermos definido de forma adequada a chave estrangeira de uma tabela e sua referência à chave primária da tabela precedente. E apesar de admitir-se, em alguns casos, a equi-junção de tabelas sem a correspondência chave primária-chave estrangeira, é recomendável não utilizar esse tipo de construção.

Seja, por exemplo, listar nome do empregado, função e nome do departamento onde ele trabalha. Observe que dois dos três dados solicitados estão na tabela “Empregado”, enquanto o outro está na tabela “Departamento”. Os dados devem ser acessados, restringindo convenientemente as relações existentes entre as tabelas. Pela lógica, “Dept_cod” é chave primária da tabela de departamentos e chave estrangeira na tabela de empregados. Portanto, este campo é o responsável pela equi-junção.

```
SELECT A.Emp_nome, A.Emp_func, B.Dept_nome
FROM Empregados A, Departamentos B
WHERE A.Dept_cod = B.Dept_cod;
```

Nota: Observe que as tabelas quando contém colunas com o mesmo nome, usa-se um apelido (*alias*) para substituir o nome da tabela associado à coluna. Imagine que alguém tivesse definido “Nome” para ser o nome do empregado na tabela de “Empregado”, e também “Nome” para ser o nome do departamento na tabela de “Departamento”. Tudo funcionaria de forma adequada, pois o *alias* se encarregaria de evitar que uma ambigüidade fosse verificada. Embora SQL resolva de forma muito elegante o problema da nomenclatura idêntica para campos de tabelas, é recomendado que se evite tal forma de nomear os campos. SQL nunca confundirá um A.NOME com um B.NOME, porém podemos afirmar o mesmo de nós mesmos? Por isto empregaremos a nomenclatura de nome de campo com um prefixo que indique o nome da tabela. Por exemplo: **Dept** para departamento + uma sublinha(_) e em seguida o campo propriamente dito; por exemplo **Dept_nome** (nome do departamento)

4.5 - Auto-Relacionamento (Self join)

O *self join* permite efetuar uma ligação de uma tabela com ela mesma. Por exemplo, numa mesma tabela (sócios) temos dois campos: um para o código de sócio propriamente dito e outro campo para o sócio que o propôs ao clube.

Socios (alias A)

Soc_codigo	Soc_nome	Soc_sexo	Soc_fone	Soc_codprop
1234-98	José da Silva	M	(44)9201-1144	2342-95
1245-99	Pedro Luiz	M	(44)9201-0035	2342-95
1342-98	Joana Braga Pereira	F	(44)9309-2345	3456-94
1453-99	Carmem da Silva	F	(44)9304-1092	3567-94

SocioProp (alias B)

Soc_codigo	Soc_nome	Soc_sexo	Soc_fone
2342-95	Antonio Carlos Silva	M	(44)9201-1191
2343-95	Luiz Carlos Souza de	M	(44)9201-9811
3456-94	Julieta Madruga	F	(44)9309-0023
3567-94	Hércules Santana	M	(44)9303-1594

Para se obter uma listagem dos sócios e respectivos proponentes deve ser feito o seguinte:

```
SELECT A.Soc_nome AS “Sócio”, B.Soc_nome AS “Proponente”
FROM Socios A, Socios B
WHERE B.Soc_codigo = A.Soc_codprop;
```

❖ **Exemplo 4.3:** Listar as matrículas de cada empregado, seu nome, seu cargo e o nome do gerente ao qual este se relaciona.

Neste caso precisamos criar um auto-relacionamento, ou seja, juntar uma tabela a ela mesma. É possível juntar uma tabela a ela mesma com a utilização de apelidos, permitindo juntar *tuplas* da tabela a outras *tuplas* da mesma tabela.

```
SELECT A.Emp_mat, A.Emp_nome, A.Emp_func, B.Emp_nome
FROM Empregados A, Empregados B
WHERE A.Emp_ger = B.Emp_mat;
```

4.6 - Sub-Consultas (*Sub queries*)

Uma sub-consulta é um comando "SELECT" que é aninhado dentro de outro "SELECT" e que devolve resultados intermediários.

Por exemplo, relacionar todos os nomes de funcionários e seus respectivos cargos, desde que o orçamento do departamento seja igual a \$30.000.

```
SELECT Emp_nome, Emp_func
FROM Empregado A
WHERE 30000 IN (SELECT Dept_orca FROM Departamento WHERE Dept_cod=A.Dept_codigo);
```

Nota: Observe que a cláusula **IN** torna-se verdadeira quando o atributo indicado está presente no conjunto obtido através da sub-consulta.

Outro exemplo: relacionar todos os departamentos que possuem empregados com remuneração superior a \$3500.

```
SELECT Dept_nome FROM Departamento A
WHERE EXISTS (SELECT *
FROM Empregado WHERE Emp_sal>3500 AND Empregado.Dept_cod = A.Dept_cod');
```

Nota: Observe que a cláusula **EXISTS** indica se o resultado de uma pesquisa contém ou não *tuplas*. Observe também que podemos verificar a não existência (**NOT EXISTS**) caso esta alternativa seja mais conveniente.

4.7 - Uniões

Podemos, eventualmente, unir duas linhas de consultas simplesmente utilizando a palavra reservada **UNION**.

❖ **Exemplo 4.4** - Listar todos os empregados que tenham códigos maiores que 10 ou funcionários que trabalhem em departamentos com código maior que 5.

O problema poderia ser resolvido com um único comando SELECT; porém, vamos empregar **UNION** para exemplificar o uso desse recurso.

```
(SELECT * FROM Empregado WHERE Emp_mat > 10)
UNION (SELECT * FROM Empregado WHERE Dept_cod>5);
```

4.8 - Inserções, Atualizações e Exclusões

Em tese, uma linguagem direcionada à extração de informações de um conjunto de dados não deveria incorporar comandos de manipulação dos dados. Contudo, devemos observar que a mera existência de uma linguagem padronizada para acesso aos dados "convida" os desenvolvedores a aderirem a uma linguagem "padrão" de manipulação de tabelas. Naturalmente, cada desenvolvedor coloca "um algo mais" em sua SQL (SQL Plus, SQL*, iSQL, e outras nomenclaturas), "desvirtuando" os objetivos da linguagem (padronização absoluta). Mas em contrapartida otimiza os acessos ao seu banco de dados e por maior que sejam estas mudanças, jamais serão tão importantes que impeçam que um programador especializado em SQL tenha grandes dificuldades em se adaptar ao padrão de determinada implementação. O fato é que as diferenças entre o SQL da Sybase, Oracle, Microsoft, são muito menores dos que as existentes entre o C, o Basic e o Pascal, que são chamadas de linguagens "irmãs", pois todas originaram conceitualmente no Fortran. Todas as três linguagens mencionadas possuem estruturas de controle tipo "para" (For), "enquanto" (While) e repita (Do..While, Repeat..Until). Todas trabalham com blocos de instruções, todas tem regras semelhantes para declaração de variáveis e todas usam comandos de tomadas de decisões baseadas em instruções do tipo "se" ou "caso"; porém, apesar de tantas semelhanças, é praticamente impossível que um programador excelente em uma linguagem consiga rapidamente ser excelente em outra linguagem do mesmo grupo: passar de Basic para C sem tramas? Quase impossível!. Poderíamos até arriscar a dizer que um excelente programador C que utilize a implementação da Symantech terá que passar por um breve período de adaptação para adaptar-se ao Turbo C da Borland. O que aconteceria então, se esse programador tivesse que adaptar-se ao Delphi (Object Pascal) da Borland? De forma alguma o mesmo ocorreria com o especialista em SQL ao ter que migrar do banco de dados **Firbird** para o banco de dados **Oracle**. Naturalmente, haveria a necessidade de um rápido aprendizado, mas este programador poderia ir se adaptando aos poucos sem precisar ser exaustivamente treinado.

4.8.1 - Inserir (INSERT)

INSERT INTO <tabela> [<campos>] [VALUES <valores>]

INSERT INTO Departamento (Dept_cod, Dept_nome, Dept_orca) **VALUES** (70, "Produção", 35000);

4.8.2 - Atualizar (UPDATE)

UPDATE <tabela> **SET** <campo> = <expressão> [WHERE <condição>];

UPDATE Empregado **SET** Emp_sal = Emp_sal* 1.2 WHERE Emp_sal< 1000;

4.8.3 - Excluir (DELETE)

DELETE FROM <tabela> [WHERE <condição>];

DELETE FROM Empregado **WHERE** Emp_sal >

4.9 - Transações

Muitas vezes gostaríamos que determinado processo, caso fosse abortado por qualquer motivo, pudesse ser inteiramente cancelado. Imaginemos por exemplo, um usuário digitando um pedido; imaginemos ainda que o sistema possa reservar cada item solicitado de maneira "on line", ou seja, ao mesmo tempo em que esta sendo digitada a quantidade

o sistema já "empenhe" uma quantidade equivalente no estoque. Imaginemos ainda que o sistema deva cancelar todas as operações se apenas um dos itens não puder ser atendido. Isto seria um grande problema caso não pudéssemos anular todos os processos a partir de determinada condição.

Vamos simular tal ocorrência na tabela "Empregado". Imaginemos que ao invés de digitar a instrução **DELETE FROM Empregados WHERE Emp_sal>5000** tivesse sido digitada a seguinte instrução: **DELETE FROM Empregados WHERE Emp_sal>500**; Neste caso, em vez de eliminarmos poucos registros, praticamente teríamos eliminado todos os registros da tabela. Para evitar que um erro de digitação ou um processo iniciado, porém sem condição de ser completado integralmente, comprometa todos os dados armazenados, pode-se criar uma transação que assegure que os testes sejam bem sucedidos ou cancelados sem comprometer outros dados.

Por exemplo, tentar excluir no máximo 20 empregados que ganham acima de 5000.

```
BEGIN TRANSACTION;
  DELETE FROM Empregado WHERE Emp_sal>5000;
  IF (SQL_RECORDCOUNT>20) THEN
    ROLLBACK;
  ELSE
    COMMIT;
  ENDIF;
END TRANSACTION;
```

Transações são usadas dentro de *stored procedures* (que serão vistas no **Capítulo 5**).

4.10 - Visões

Uma visão consiste basicamente de uma tabela derivada de outras tabelas.

Considerando o nosso banco de dados, poderíamos criar uma visão baseada na tabela "Empregado" e na tabela "Departamento", onde tivéssemos somente os nomes dos empregados e os departamentos nos quais eles trabalhassem. Teríamos algo semelhante ao mostrado abaixo.

```
CREATE VIEW EmpDep
AS SELECT E.Emp_nome, D.Dept_nome
FROM Empregado E, Departamento D
WHERE E.Dept_cod = D.Dept_cod;
```

Dependendo do banco de dados e sua configuração, essa sintaxe pode sofrer algumas alterações; por exemplo, aspas entre os nomes dos campos e da tabela pode ser uma delas.

Deve ser observado que:

- 1) Uma visão definida sobre uma única tabela somente será atualizável se os atributos da tal visão contiverem a chave primária de tal tabela.
- 2) Visões sobre várias tabelas não são passíveis de atualizações.
- 3) Visões que utilizam funções de agrupamentos não poderão ser atualizadas.

4.11 - Exemplos de instruções sql

Para os exercícios a seguir, considere um banco de dados denominado "Empresa", com as seguintes tabelas:

- **Departamento**

Dept_codigo código do departamento
 Dept_nome nome do departamento
 Dept_orca orçamento do departamento
 Dept_loca localização do departamento na empresa

- **Empregado**

Emp_mat matrícula do empregado
 Emp_nome nome do departamento
 Emp_sexo sexo do empregado
 Emp_func função do empregado no departamento
 Emp_sal salário do empregado
 Emp_admis data de admissão do empregado
 Emp_comi comissão do empregado

- **Clientes**

Cli_codigo código do cliente
 Cli_nome nome do cliente
 Cli_endereco endereço do cliente
 Cli_cid cidade do cliente

- **Fornecedor**

For_codigo código do fornecedor
 For_nome nome do fornecedor
 For_endereco endereço do fornecedor

- **Produto**

Prod_codigo código do produto
 Prod_nome nome do produto
 Prod_preco preço do produto

1) Seleção de todos os campos (colunas) da tabela de departamentos.

SELECT * FROM Departamento

O exemplo utiliza o coringa "*" para selecionar as colunas na ordem em que foram criadas. A instrução **SELECT**, seleciona (extrai) um grupo de registros de uma (ou mais) tabela(s). No exemplo a cláusula **FROM** indica a necessidade de pesquisarmos tais dados apenas na tabela "Departamento".

2) Selecione todos os departamentos cujo orçamento mensal seja maior que \$100.000. Apresente os nomes dos departamentos com seus respectivos orçamentos anuais.

Neste caso precisamos de uma expressão que é a combinação de um ou mais valores, operadores ou funções que resultarão em um único valor. Essa expressão poderá conter nomes de colunas, valores numéricos, constantes e operadores.

SELECT Dept_nome, Dept_orca **FROM** Departamento **WHERE** Dept_orca>100000;

3) Apresente a instrução anterior, porém ao invés de "Dep_nome" e "Dep_orca", os títulos deverão ser "Departamento" e "Orçamento", respectivamente.

Neste exemplo deveremos denominar as colunas por apelidos. Os nomes das colunas mostradas por uma consulta são geralmente os nomes existentes no Dicionário de Dados; porém, geralmente estão armazenados na forma do mais puro "informatiquês", onde "todo mundo" sabe que "Cli_cod" significa "código do Cliente". Mas é possível (e provável) que o usuário desconheça essas expressões; portanto devemos renomear com apelidos as colunas "contaminadas" pelo "informatiquês", que apesar de fundamental para os analistas somente são vistos como enigmas para os usuários.

```
SELECT Dept_nome AS "Departamento", Dept_orca AS "Orçamento"
FROM Departamento
WHERE Dept_orca>100000;
```

4) Apresente todas as funções existentes na empresa, porém omita eventuais duplicidades.

A cláusula **DISTINCT** elimina duplicidades, significando que somente relações distintas serão apresentadas como resultado de uma pesquisa.

```
SELECT DISTINCT Emp_func FROM Empregado;
```

5) Apresente todos os dados dos empregados, considerando sua existência física diferente de sua existência lógica.

Desejamos um tratamento diferenciado para valores nulos. Qualquer coluna de uma *tupla* que não contenha informações é considerada **nula**; portanto informação não existente. Isto não é o mesmo que "zero", pois zero é um número como outro qualquer, enquanto que um valor nulo (NULL) utiliza um *byte* de armazenagem interna e são tratados de forma diferenciada pela SQL.

```
SELECT Emp_mat, Emp_nome, Emp_sexo, Emp_func, Emp_sal, Emp_comi
FROM Empregado;
```

```
SELECT Emp_mat, Emp_nome, Emp_sexo, Emp_func, NVL(Emp_sal,0) + NVL(Emp_comi,0)
FROM Empregado;
```

Nota: A função "NVL" é utilizada para converter valores nulos em zeros.

6) Apresente os nomes e funções de cada funcionário contidos na tabela empresa, porém classificados alfabeticamente (A..Z) e depois alfabeticamente invertido (Z..A).

A cláusula **ORDER BY** modificará a ordem de apresentação do resultado da pesquisa (ascendente ou descendente).

```
SELECT Emp_nome, Emp_func
FROM Empregado
ORDER BY Emp_nome;
```

```
SELECT Emp_nome, Emp_func
FROM Empregado
ORDER BY Emp_nome DESC;
```

Nota: Também é possível fazer com que o resultado da pesquisa venha classificado por várias colunas. Sem a cláusula **"ORDER BY"** as linhas serão exibidas na sequência que o SGBD determinar.

7) Selecione os nomes dos departamentos que estejam na fábrica de São Paulo.

```
SELECT Dept_nome FROM Departamento WHERE Dept_loca = "São Paulo";
```

O exemplo exigiu uma restrição (São Paulo) que nos obrigou a utilizar da cláusula **WHERE**. Alguns analistas costumam afirmar em tom de brincadeira que SQL não passa de *"selecione algo de algum lugar onde se verificam tais relações"*.

8) Selecione os empregados cujos salários sejam menores que 1000 ou maiores que 3500.

Aqui será necessária a utilização de expressão com o operador lógico **NOT**.

```
SELECT Emp_nome, Emp_sal FROM Empregado
WHERE Emp_sal NOT (BETWEEN 1000 AND 3500);
```

9) Apresente todos os funcionários com salários de 800 a 1600 e que sejam vendedores.

```
SELECT Emp_nome, Emp_sal, Emp_func FROM Empregado
WHERE Emp_sal BETWEEN (800 AND 1600)
AND Emp_func = "Vendedor";
```

10) Apresente todos os empregados com seus respectivos departamentos, ordenados pelos seus nomes, dentro de cada departamento.

```
SELECT Emp_nome, Dept_nome FROM Empregado E, Departamento D
WHERE (D.Dept_codigo=E.Dept_codigo)
ORDER BY Dept_nome, Emp_nome';
```

11) Apresente todos os empregados com salários entre 800 e 1600 e que sejam vendedores ou balconistas.

```
SELECT Emp_nome, Emp_sal, Emp_func FROM Empregado, Departamento
WHERE (Empregado.Dept_codigo=Departamento.Dept_codigo
AND Emp_func = "Balconista" OR Emp_func = "Vendedor" );
```

12) Apresente o nome de todos os empregados em letras minúsculas.

```
SELECT LOWER( Emp_nome ) FROM Empregado;
```

13) Apresente o nome de todos os empregados (somente as 10 primeiras letras).

```
SELECT SUBSTRING (1, 10, Emp_nome) FROM Empregado;
```

14) Apresente a média, o maior, o menor e também o somatório dos salários pagos aos empregados.

```
SELECT AVG(Emp_sal) FROM Empregado;
SELECT MIN(Emp_sal) FROM Empregado;
SELECT MAX(Emp_sal) FROM Empregado;
SELECT SUM(Emp_sal) FROM Empregado;
```

15) Apresente a média dos orçamentos por departamento.

```
SELECT Dept_nome, AVG(Emp_sal) FROM Departamento, Empregado
WHERE Departamento.Dept_codigo=Empregado.Dept_codigo GROUP BY Dept_nome;
```

16) Retome o problema anterior, porém apresente resposta apenas para departamentos com mais de 10 empregados.


```
SELECT Dept_nome, AVG(Emp_sal) FROM Departamento, Empregado
WHERE Departamento.Dept_codigo=Empregado.Dept_codigo GROUP BY Dept_nome
HAVING COUNT(*)>10;
```

Nota: A cláusula **GROUP BY** deve ser colocada antes da **HAVING**, pois os grupos são formados e as funções de grupos são calculadas antes de se resolver a cláusula **HAVING**.
A cláusula "**WHERE**" não pode ser utilizada para restringir grupos que deverão ser exibidos.

{ Seleção ERRADA }

```
SELECT Dept_nome, AVG(Emp_sal) FROM Empregado WHERE AVG(Emp_sal) > 1000
GROUP BY Dept_nome;
```

{Seleção CORRETA}

```
SELECT Dept_nome, AVG(Emp_sal) FROM Empregado GROUP BY Dept_nome
HAVING AVG(Emp_sal) > 1000;
```

A sequência correta num comando "SELECT" é a seguinte:

SELECT	coluna(s)
FROM	tabela(s)
WHERE	condição(ões) da(s) tupla(s) individualmente
GROUP BY	condição(ões) do(s) grupo(s) de tupla(s)
HAVING	expressão para selecionar grupos de tuplas
ORDER BY	coluna(s);

A instrução *sql*/ fará a seguinte avaliação:

- a) WHERE: para estabelecer *tuplas* individuais candidatas (não pode conter funções de grupo).
- b) GROUP BY: para fixar grupos.
- c) HAVING: para selecionar grupos na exibição.

17) Selecionar apenas os campos código, nome e telefone da tabela "Fornecedor".

```
SELECT For_codigo, For_nome, For_fone FROM Fornecedor;
```

18) Selecionar os campos código e nome na tabela "Clientes", cujos clientes possuam código maior que 10 e menor que 100.

```
SELECT Cli_codigo FROM Clientes WHERE (Cli_codigo>10 AND Cli_codigo<100);
```

19) Selecionar os campos cidade e nome da tabela "Clientes", agrupando-os por esses dois campos.

```
SELECT Cli_cid, Cli_nome FROM Clientes GROUP BY Cli_cid, Cli_nome;
```

20) Considerando as tabelas "Fornecedor" e "Produto" numa relação 1:N e que o campo comum entre elas é o campo For_codigo, selecionar todos registros desse relacionamento.

```
SELECT * Fornecedor, Produto WHERE Fornecedor.For_codigo=Produto.For_codigo;
```

21) Selecionar todos os clientes cujo nome começa com a letra "J"

```
SELECT * FROM Clientes WHERE Cli_nome LIKE "J%";
```

22) Atualizar o campo salário da tabela “Folha”, aumentando em 20%, todos os funcionários lotados no setor “DIFIN”

UPDATE Folha **SET** Folha_salario=Folha_salario*1.2 **WHERE** setor = "DIFIN";

23) Excluir da tabela “Funcionário”, todos aqueles que ainda são estagiários.

DELETE FROM Funcionario **WHERE** Fun_status="ESTAGIÁRIO";

24) Selecionar todos os títulos da tabela “Titulo”, com vencimento em 30/11/2009.

SELECT * FROM Titulo **WHERE** vencimento='30/11/2009'

25) Selecionar todos os funcionários cujo salário que varie de 1500 a 2500.

SELECT * FROM Folha **WHERE** Folha_salario **BETWEEN** (1500 **AND** 2500);

26) Selecionar os clientes que residem num dos seguintes estados: MG, AP, RJ, PR ou BA

SELECT * FROM Cliente **WHERE** Cli_estado **IN** ("MG", "RJ", "AP", "PR", "BA");

27) Somar todos os salários da tabela “Folha” e mostrar esse total em uma coluna denominada “TotalSalarios” .

SELECT SUM(Folha_salario) **AS** TotalSalarios **FROM** Folha

Obs): O campo “TotalSalarios” criado no Exemplo 12 é virtual; não será inserido na tabela.

28) Considerando as tabelas “Livro” e “Autor” (mostradas a seguir), listar os campos Liv_titulo e Aut_nome.

SELECT Livro.Liv_titulo, Autor.Aut_nome **FROM** Livro, Autor **WHERE** Autor.Aut_id=Livro.Aut_id

Em termos de relacionamento entre entidades, o atributo Aut_id é chave primária em “Autor” e chave estrangeira em “Livro”. Isto quer dizer que um autor pode ter vários livros, e um livro (neste caso) só pode ter um autor.

29) Considerando a tabela “Clientes” abaixo, com alguns dados, listar as cidades, sem repetições.

Cli_id	Cli_nome	Cli_cidade	Cli_estado	Cli_fone
1	Abel Sharon Goldman	Juazeiro	BA	(74)222-3344
2	Ali Mohamed Kaled	Tombos	MG	(32)259-2266
3	Jamal El-Kaled Lotar	Natividade	RJ	(24)245-3344
4	Ariel Samou Siate	Botacatu	SP	(14)234-2211
5	Dalila Kalil Gebara	Petrópolis	RJ	(22)266-4422
6	Samira Ali-Bachar	Foz do Iguaçu	PR	(45)544-5533
7	Sara Goldman Levi	Foz do Iguaçu	PR	(45)533-6655

SELECT DISTINCT Cli_cidade **FROM** Clientes;

30) Considerando as três tabelas a seguir: “Autor”, “Editora” e “Livro”, liste as editoras com os respectivos totais de livros:

Autor

Aut_id	Aut_nome
1	Cristiane Gonçalves
2	Wilson de Pádua de Paula Filho
3	Mário Leite
4	Carlos A. J. Oliviero
5	James A. O'Brien
6	Douglas Hergert
7	Fernando de Souza Meirelles
8	Douglas Hergert
9	Álvaro Luíz Arouche C. Ramos
10	Antônio Geraldo da Rocha Vidal

Editora

Edit_id	Edit_nome
1	LTC - Livros Técnicos e Científicos Editora
2	Érica
3	Ciência Moderna
4	Campus
5	Saraiva

Livro

Edit_id	Liv_titulo	Aut_nome
3	BrOffice.org Calc Avançado com Introdução às Macros	Cristiane Gonçalves
1	Multimídia: Conceitos e Aplicações,	Wilson de Pádua de P. Filho
2	Faça um site – Dreamweaver Orientado por Projeto	Carlos A. J. Oliviero
5	Sistemas de Informação	James A. O'Brien
1	Análise Estruturada de Sistemas	Chris Gane
1	Gerando Gráficos em Clipper com C	Álvaro Luíz Arouche C. Ramos
3	SciLab: Uma Abordagem Prática e Didática	Mário Leite
1	Clipper – Versão Summer 87 Volume 2	Antônio Geraldo da Rocha Vidal
4	Introdução ao Turbo Basic	Douglas Hergert

```

SELECT Editora.Edit_nome, COUNT(Livro.Edit_id) AS TotalLivros
FROM Editora INNER JOIN Livro ON Editora.Edit_id = Livro.Edit_id
GROUP BY Editora.Edit_nome

```

Editora	TotalLivros
Livros Técnicos e Científicos	4
Érica	1
Makron Books	1
Campus	1
Saraiva	1
CiênciaModerna	2

31) Considerando, ainda, as três tabelas do exemplo anterior, listar somente os registros com as editoras que possuam mais de um livro.

```

SELECT Editora.Edit_nome, COUNT(Livro.Edit_id) AS TotalLivros
FROM Editora INNER JOIN Livro ON Editora.Edit_id = Livro.Edit_id
GROUP BY Editora.Edit_nome
HAVING COUNT(Livro.Edit_id)>1

```

32) Dadas duas tabelas: uma de funcionários e outra de dependentes, com o campo “Func_mat” comum às duas; selecionar todos os funcionários com os respectivos dependentes. Por exemplo, o funcionário “Manoel Pereira dos Santos”, cuja matrícula é 22765-0, tem 4 dependentes. Listar todos os funcionários com seus respectivos dependentes.

```
SELECT Funcionario.Func_nome, Dependente.Dep_nome
FROM Funcionario, Dependente
ON Funcionario.Func_mat = Dependente.Func_mat
```

33) Considere uma consulta que associe as tabelas “Clientes” e “Pedidos” no campo Cli_codigo. A tabela “Clientes” não contém campos “Cli_codigo” duplicados, mas a tabela “Pedidos” sim, pois cada cliente pode ter vários pedidos. A instrução *sql* abaixo mostra como podemos produzir uma listagem das empresas que tenham pelo menos um pedido, usando o *predicado* **DISTINCTROW**.

```
SELECT DISTINCTROW Cli_nome FROM Clientes, Pedidos
WHERE Clientes.Cli_codigo=Pedidos.Cli_codigo ORDER BY Cli_nome
```

O Predicado **DISTINCTROW** omite dados com base em registros inteiramente duplicados e não somente campos duplicados, mas sem qualquer detalhe sobre os pedidos, contidos na instrução: **SELECT DISTINCTROW** Cli_nome

34) Aumentar em 15% o salário dos funcionários, da tabela “Folha”, que são estão ganhando atualmente um salário menor que \$1.000.

```
UPDATE Folha SET Folha_salario=Folha_salario*1.15 WHERE Folha_salario<1000
```

35) Selecionar todos os empregados com salários de 800 a 2000 e que sejam vendedores ou motoristas.

```
SELECT Emp_nome FROM Empregado WHERE Emp_sal BETWEEN (800 AND 2000)
AND (Emp_cargo="Vendedor" OR Emp_cargo="Motorista")
```

36) Extrair os nomes de todos os empregados, porém com letras maiúsculas.

```
SELECT UPPER(Emp_nome) FROM Empregado
```

37^[12] Listar a matrícula de cada empregado, o nome, o cargo e o nome do gerente ao qual está subordinado, sabendo que cada gerente pode ter vários subordinados, mas cada empregado está subordinado a um único gerente.

```
SELECT A.Emp_matr,A.Emp_Nome,E.Emp_cargo, B.Emp_nome,
FROM Empregado A, Empregado B WHERE A.Emp_cod = B.Emp_cod
```

38) Exibir os registros dos empregados com o campo salário acrescido de um aumento de 15% e renomeado (virtualmente) como NovoSalario

```
SELECT Emp_nome, Emp_sal*1.15 AS NovoSalario FROM Empregado ORDER BY Emp_nome
```

39^[13]) Lista todos os empregados com seus respectivos cargos para os departamentos cujo orçamentos sejam maiores que 50000.

¹² Neste exemplo temos um Auto-relacionamento. Devemos juntar uma tabela a ela mesma, pois gerente é também um empregado. Isto é possível com o empregado dos apelidos **A** e **B**. para a mesma tabela.

```
SELECT Emp_nome, Emp_cargo FROM Empregado A
WHERE 50000 IN (SELECT Dept_orca FROM Departamento
WHERE Dept_codigo = A.Dept_codigo)
```

40) Lista todos os departamentos que possuem empregados com salário superior a 5000.

```
SELECT Dept_nome FROM Empregado A
WHERE EXISTS (SELECT * FROM Empregado
WHERE Emp_salario>5000 AND Dept_codigo = A.Dept_codigo)
```

4.12 - Exercício proposto

Crie o MER e o banco de dados com as relações abaixo; em seguida normalize-o. Os pontos de interrogação (?) representam outro(s) atributo(s) caso você ache que sejam necessários.

- **Cliente** (Cli_codigo, Cli_nome, Cli_sexo, Cli_endereco, Cli_cidade, Cli_estado, Cli_cep, Cli_fone, Cli_celular, Cli_email, Cli_cpf, Cli_rg, ?)
- **Vendas** (Venda_num, Venda_data, Venda_valor, ?)
- **TipoVenda** (TipoVenda_cod, TipoVenda_desc, ?)
- **Produto** (Prod_cod, Prod_desc, Prod_unid, Prod_preco, ?)
- **TipoProd** (TipoProd_cod, TipoProd_desc, ?)
- **ItensVenda** (Item_num, Item_quant, Item_total, ?)
- **ContasReceber** (Conta_num, Conta_data, Conta_vencimento, ?)
- **Vendedor** (Vendo_mat, Vendo_nome, Vendo_sexo, Vendo_fone, Vendo_email, ?)

Faça uma análise do banco, e considere as restrições que você achar necessárias.

¹³ Este exemplo (e o 39) são casos de sub-consulta, onde um comando SELECT está aninhado dentro de outro SELECT. Entretanto, quando uma subconsulta ler dados de mais de uma tabela, devemos usar a Equi-Junção.

Capítulo 5 - Servidores de Bancos de Dados

5.1 - Introdução

A abordagem feita no Capítulo 2 sobre “bancos de dados”, abrangeu o aspecto físico e o gerenciamento de maneira teórica; como um SGBD controla os dados e como esses dados formam módulos que os armazenam, as tabelas e todos os outros elementos necessários para um bom gerenciamento desses dados armazenados. Este capítulo focalizará, mesmo que de forma introdutória, os Servidores de Bancos de dados: os *softwares* que, efetivamente, armazenam e gerenciam os dados de forma adequada. Existe hoje em dia no mercado uma gama enorme de servidores de bancos de dados; entre eles podem ser citados: Interbase, Firebird, MySQL, DB2 SQL Server, PostgreSQL, Sybase, Access, etc. A classificação desses servidores pode ser quanto ao formato de armazenamento de seus dados (arquivo único ou arquivos separados) ou ainda em corporativos e *desktops*. Na verdade, as diversas classificações que podem existir não interessam muito, pois o que realmente importa é se o servidor escolhido atende as necessidades da empresa e se estão de acordo com a estratégia gerencial da organização. Os dois primeiros citados (Interbase e Firebird) são quase como irmãos gêmeos, pois o Firebird nasceu de “uma costela” do Interbase. Assim, desde o seu nascimento, Firebird é quase 100% compatível com o Interbase, apesar de ninguém poder apostar até quando essa compatibilidade vai durar. A verdade é que essa compatibilidade é tanta que na ferramenta de desenvolvimento Delphi existe um conjunto de componentes denominado “Interbase” (para acesso a bancos do Interbase) que é plenamente utilizada em acessos a bancos do Firebird; somente na próxima versão do Delphi (Delphi 2010) é que haverá um *drive (dbExpress)* próprio de acesso a bancos Firebird. A vantagem do Firebird sobre o Interbase é que enquanto o Interbase é um SGBD proprietário (da Borland), o Firebird é *free*, e pode ser baixado da Internet e utilizado livremente sem pagamento de *royalties*. Neste trabalho será estudado o Firebird, pois além de ser *free* é bem parecido com os outros servidores, embora outros citados possam ser mais robustos, como por exemplo, DB2 SQL Server, e Oracle. Entretanto, o Firebird é dos mais utilizados pelos programadores de *software* para as empresas de pequeno e médio porte, e isto é importantíssimo, pois no Brasil elas representam mais de 85% do total das organizações, gerando a maioria dos postos de trabalho do país. Além do mais, conforme foi enfatizado no Capítulo 1, o SGBD escolhido pela empresa não precisa ser necessariamente, “top de linha”, e sim aquele que melhor atenda suas necessidades.

5.2 - O Firebird

5.2.1- Download e Instalação

Conforme explicado, o Firebird foi criado a partir do Interbase, com 100% de compatibilidade até as versões Interbase 6.5 e Firebird 2.0 (um ou outro instalado numa máquina; os dois juntos podem ocorrer conflitos); entretanto, a tendência é o Firebird seguir o seu próprio caminho e se tornar um SGBD fortíssimo para a concorrência. Neste trabalho será utilizada a versão 3.05 (a versão oficial atual mais estável do produto), embora já exista a versão 4.0 em fase *alfa* (com testes iniciais). Isto quer dizer que todas

as outras versões anteriores: 1.0, 1.3, 1.5 e 2.0, 3.x também podem ser utilizadas para gerenciar bancos de dados, pois são estáveis, embora possam conter alguns pequenos *bugs*, o que é perfeitamente natural.

Para baixar o servidor Firebird é aconselhável fazer isto a partir do *site* oficial da “empresa” que o gerencia - *Comunidade Firebird* - deve ser acessado o *link*: <http://www.firebirdsql.org/index.php?op=files&id=engine> (acesso em 20/05/2020 - 13:08). A **figura 5.1a** mostra a página oficial do produto; clicando na versão desejada a página da **figura 5.1b** será exibida para executar o *download*.

Firebird True universal open source database

HOME DOCUMENTATION **DOWNLOADS** COMMUNITY SUPPORT DEVELOPMENT

You are here: Home > Downloads > Server Packages

Server Packages

- Firebird 3.0
- Firebird 2.5
- Snapshot Builds
- Discontinued Versions
- Firebird 4.0 Beta 2 (unstable)

Connectivity

- JDBC Driver
- .NET Provider
- Additional Downloads
- Python Driver
- ODBC Driver
- PHP Driver
- Firebird Butler
- Other Downloads
- Sphinx Full Text Search
- Third-party Tools & Drivers

Server Packages

Major Release	Latest Version	Release Date	Release Notes
Firebird 3.0	3.0.5	9 Jan 2020	
Firebird 2.5	2.5.9	24 Jun 2019	

Discontinued versions can be found [here](#).

[Join Firebird!](#) [Tweet](#) [Share](#) [RSS](#) [Email](#)

Figura 5.1a - Página oficial para baixar o Firebird

HOME DOCUMENTATION **DOWNLOADS** COMMUNITY SUPPORT DEVELOPMENT

You are here: Home > Downloads > Server Packages > Firebird 3.0 > Firebird 3.0.5

Server Packages

- Firebird 3.0
- Firebird 2.5
- Snapshot Builds
- Discontinued Versions
- Firebird 4.0 Beta 2 (unstable)

Connectivity

- JDBC Driver
- .NET Provider
- Additional Downloads
- Python Driver
- ODBC Driver
- PHP Driver

Firebird Butler

Other Downloads

- Sphinx Full Text Search
- Third-party Tools & Drivers

Firebird 3.0.5

Press Release: [English](#) | [Русский](#) | [Português brasileiro](#)

The primary goals for Firebird 3.0 were to unify the server architecture and to improve support for SMP and multiple-core hardware platforms. Parallel objectives were to improve threading of engine processes and the options for sharing page cache across thread and connection boundaries.

Alongside these aims came new strategies to improve performance, query optimization, monitoring and scalability and to address the demand for more security options. A number of popular features were introduced into the SQL language, including the long-awaited support for the Boolean data type and the associated logical predications.

Documentation: [Release Notes](#) (PDF available), [Quick Start Guide](#) (PDF available) and [other manuals](#).

Win32 Win64 Linux x86 Linux AMD64 Android Mac OS X

Release Date	File Name	Size	Description
Sources			
January 9, 2020	Firebird-3.0.5.33220-0.tar.bz2	9 MB	Compressed tarball
Win32			
32-bit Kits			
January 9, 2020	Firebird-3.0.5.33220_0_Win32.exe	7 MB	Windows executable installer, recommended for first-time users
January 9, 2020	Firebird-3.0.5.33220-0_Win32.zip	12 MB	Zip kit for manual/custom installs

Join Firebird!

Figura 5.1b - Selecionando a versão do produto para o download...

Clicando no *link* do *setup* (exe) uma janelinha, como a da **Figura 5.1c**, se apresentará, informando que o sistema está pronto para iniciar o *download* do servidor.

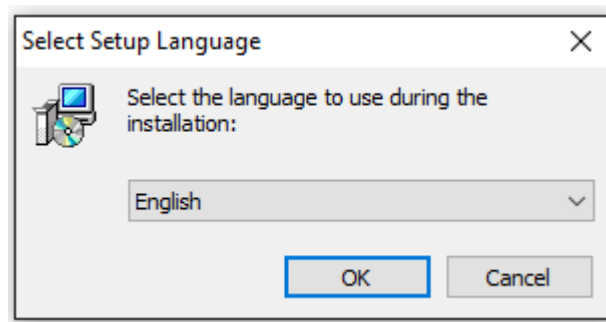


Figura 5.1c - Janelinha de download do Firebird

Escolhendo o idioma, e conformando no botão **[OK]** aparece a janela da figura 5.1d para aceitar as condições da licença.

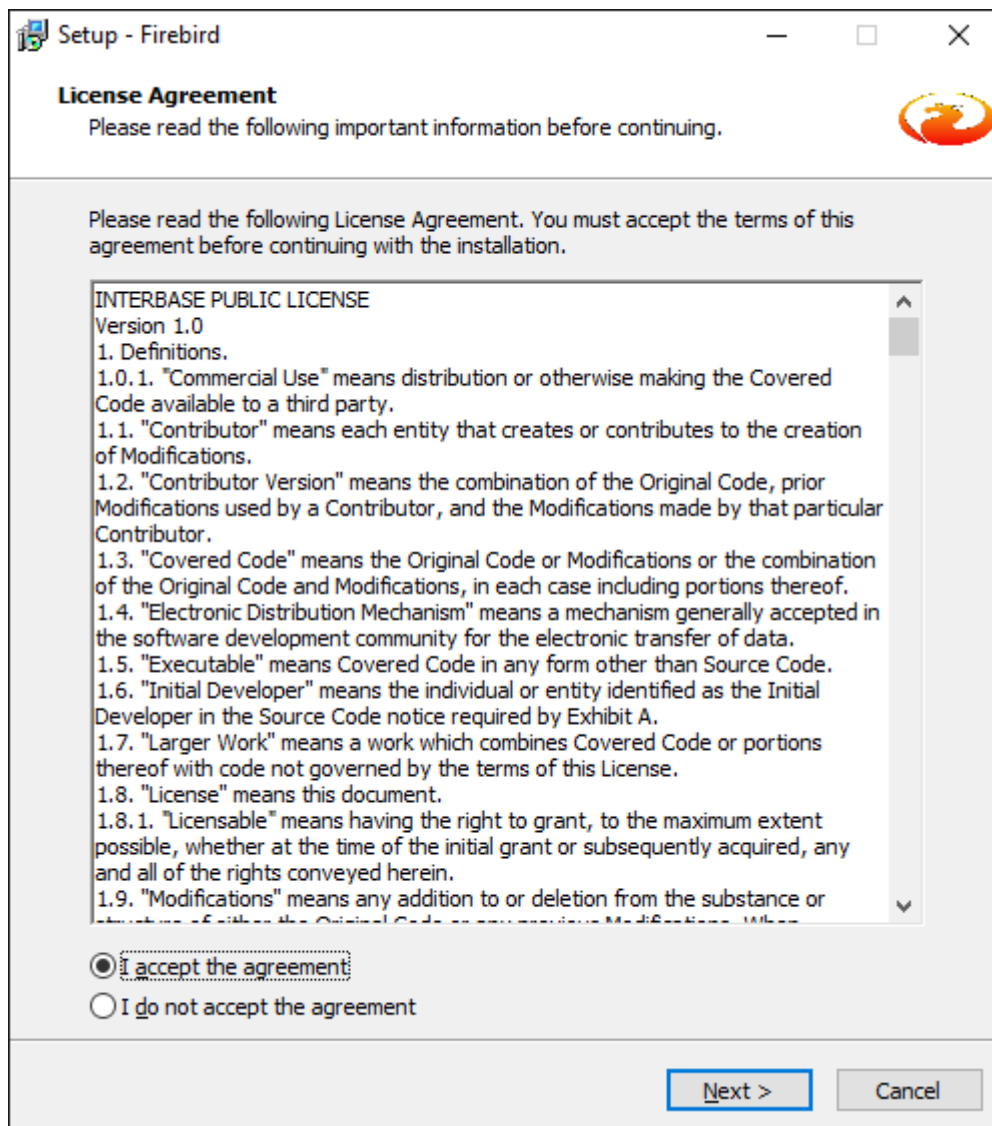


Figura 5.1d - Janela de aceitação da licença

Clicando no botão **[Next >]** da janela da **figura 5.1d** o processo de instalação avança, e aparece a janela da **figura 5.1e**, com informações pertinentes a respeito da versão do Firebird ser instalada.



Figura 5.1e - Janela de com informações a Instalação

Clicando no botão **[Next >]** da janela da **figura 5.1e** o processo avança para mostrar a janela da **figura 5.1f**, para definir o local da instalação.

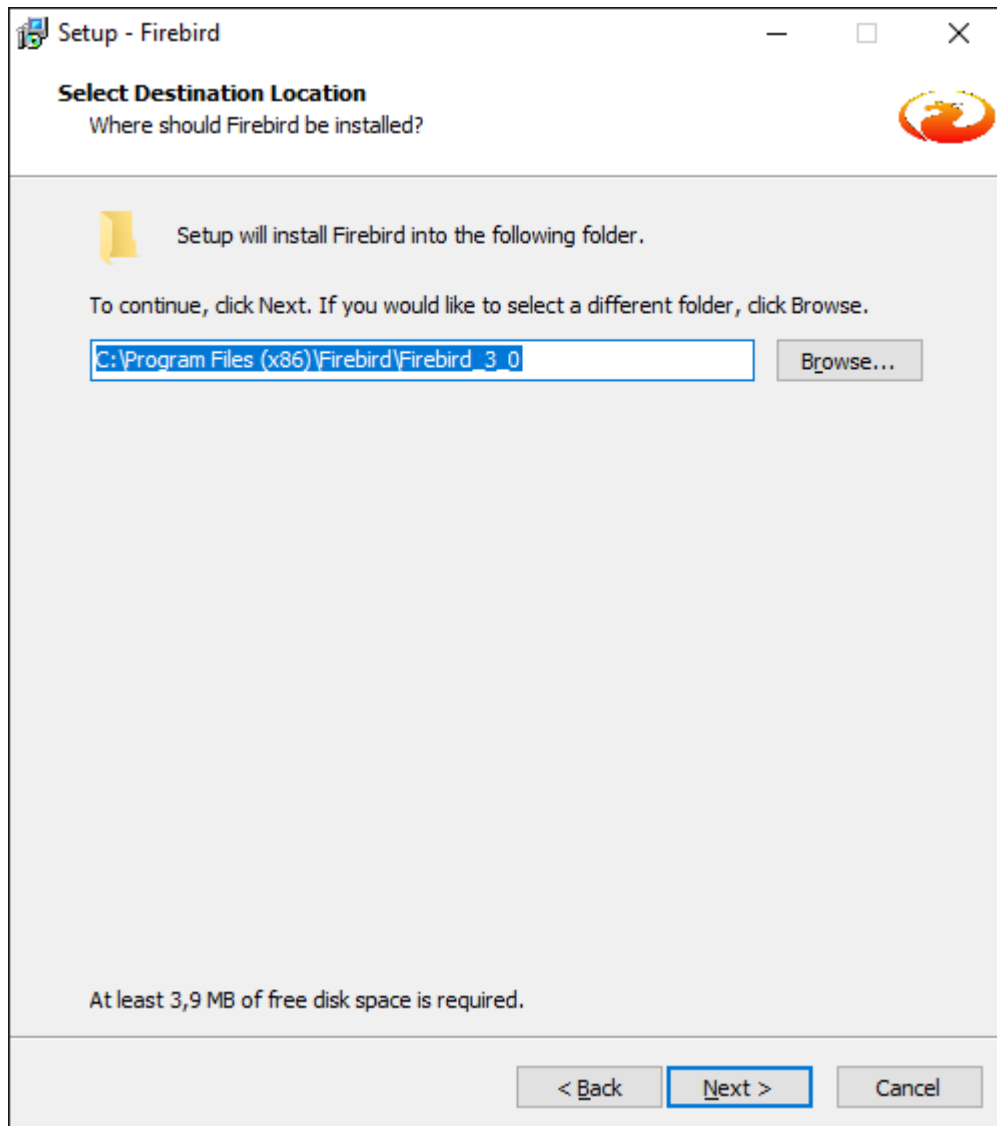


Figura 5.1f - Janela de definição do local de instalação

A menos que o usuário tenha alguma razão bem específica, é sugerido que se considere o local sugerido (padrão) indicado na tela da **figura 5.1f**.

Assim, clicando no botão **[Next >]** da janela da **figura 5.1f**, o processo apresenta a janela da **figura 5.1g** para ser selecionado o componente a ser instalado. A sugestão é que deixe marcados os apresentados.

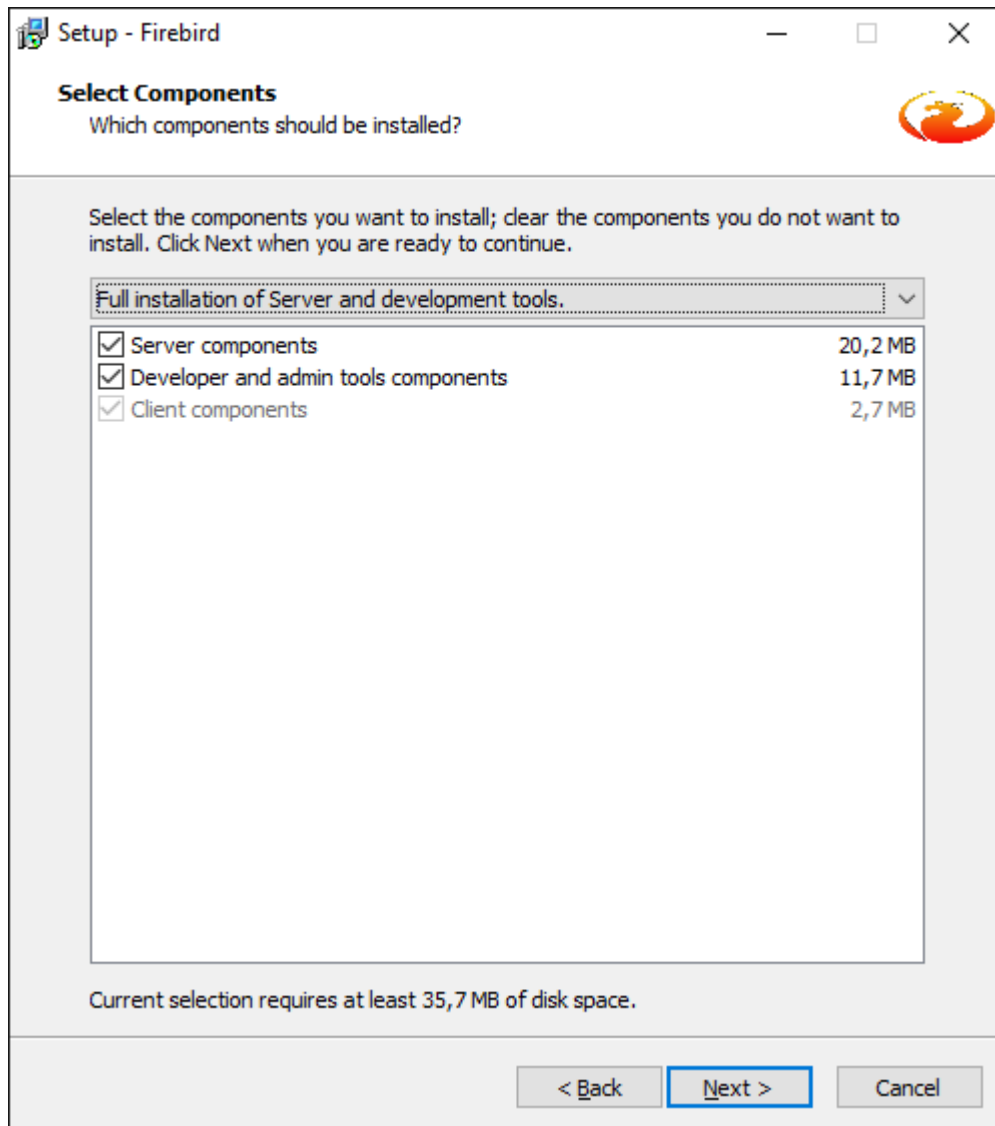


Figura 5.1g - Janela de seleção dos componentes

Clicando no botão **[Next >]**, depois de selecionar os componentes, aparece a janela da **figura 5.1h** para que o usuário selecione os componentes a serem instalados. Marque, de acordo com a figura, não se esquecendo dos itens que tratam das bibliotecas. O tipo de arquitetura da ferramenta (*Classic*, *Server* e *Superserver*), depende de como se quer o Firebird trabalhando sob o Windows; mas, não se esqueça de marcar as opções de bibliotecas.

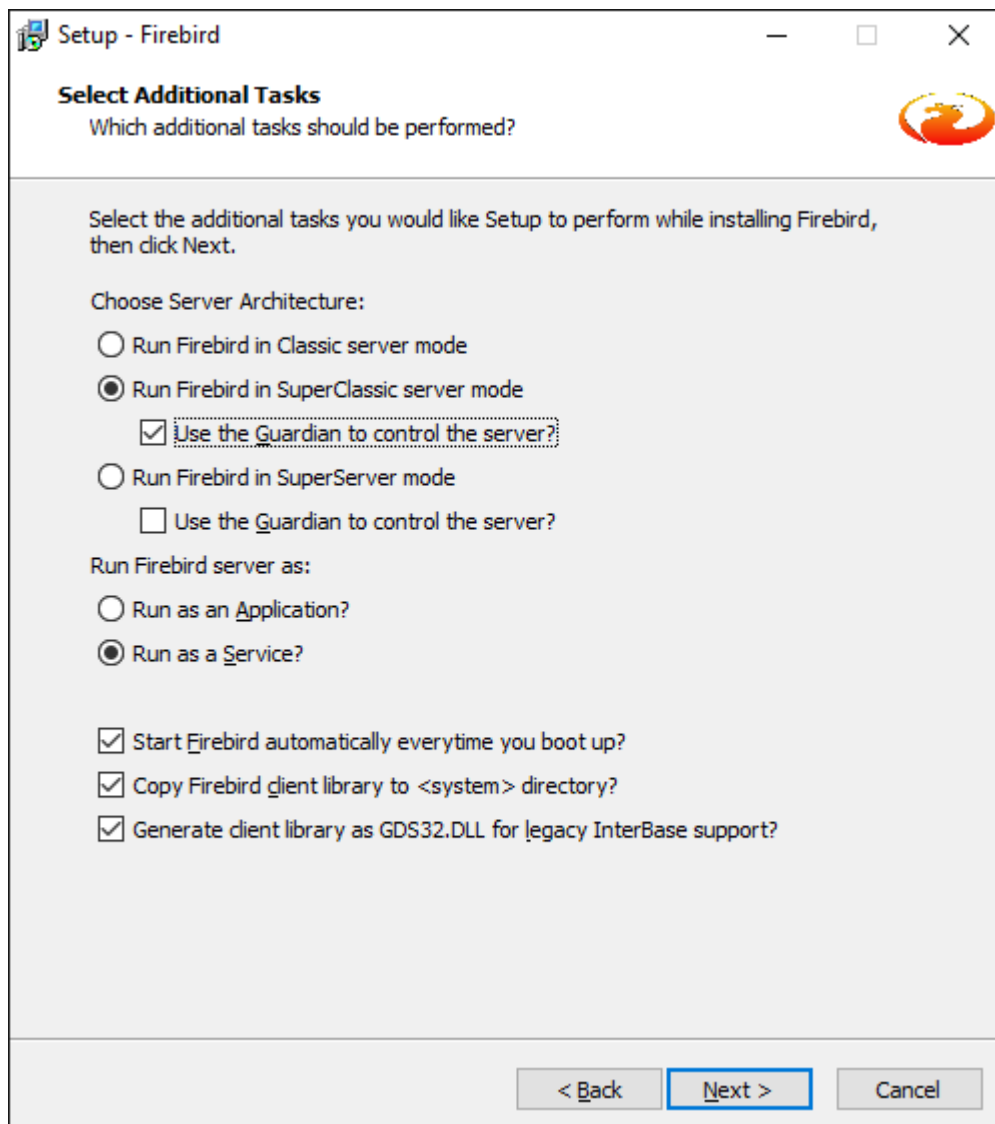


Figura 5.1h - Janela de seleção da arquitetura do servidor

Clicando no botão **[Next >]** aparece a janela da **figura 5.1i**, informando que as configurações foram feitas com sucesso, e que o sistema está pronto para ser instalado...

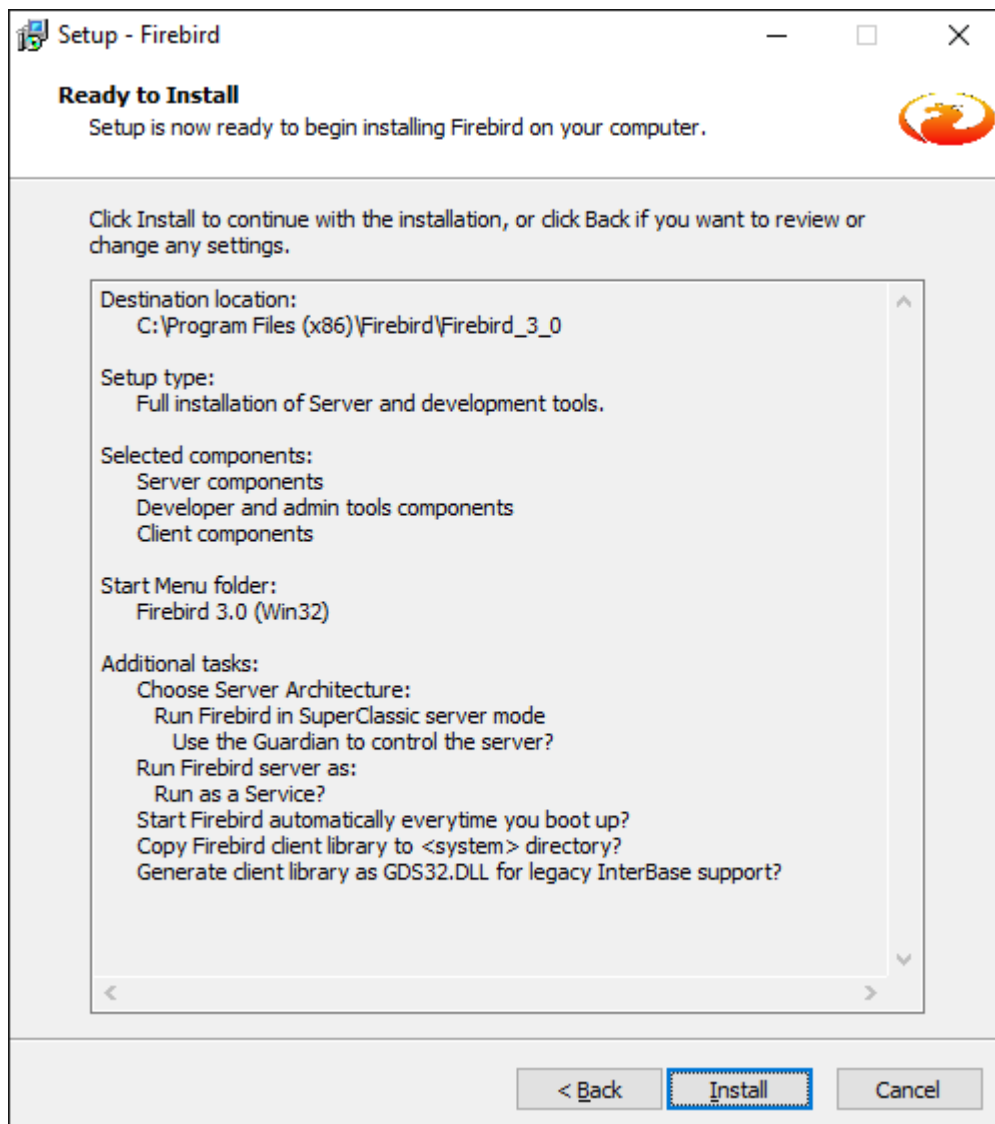


Figura 5.1i - Janela para informar que está pronto para instalação

Finalmente, na janela da **figura 5.1j** clique no botão **[Install]** [ara que o programa seja instalado, de acordo com as configurações dadas nos passos anteriores.

A janela da **figura 5.2a** mostra o progresso da instalação.

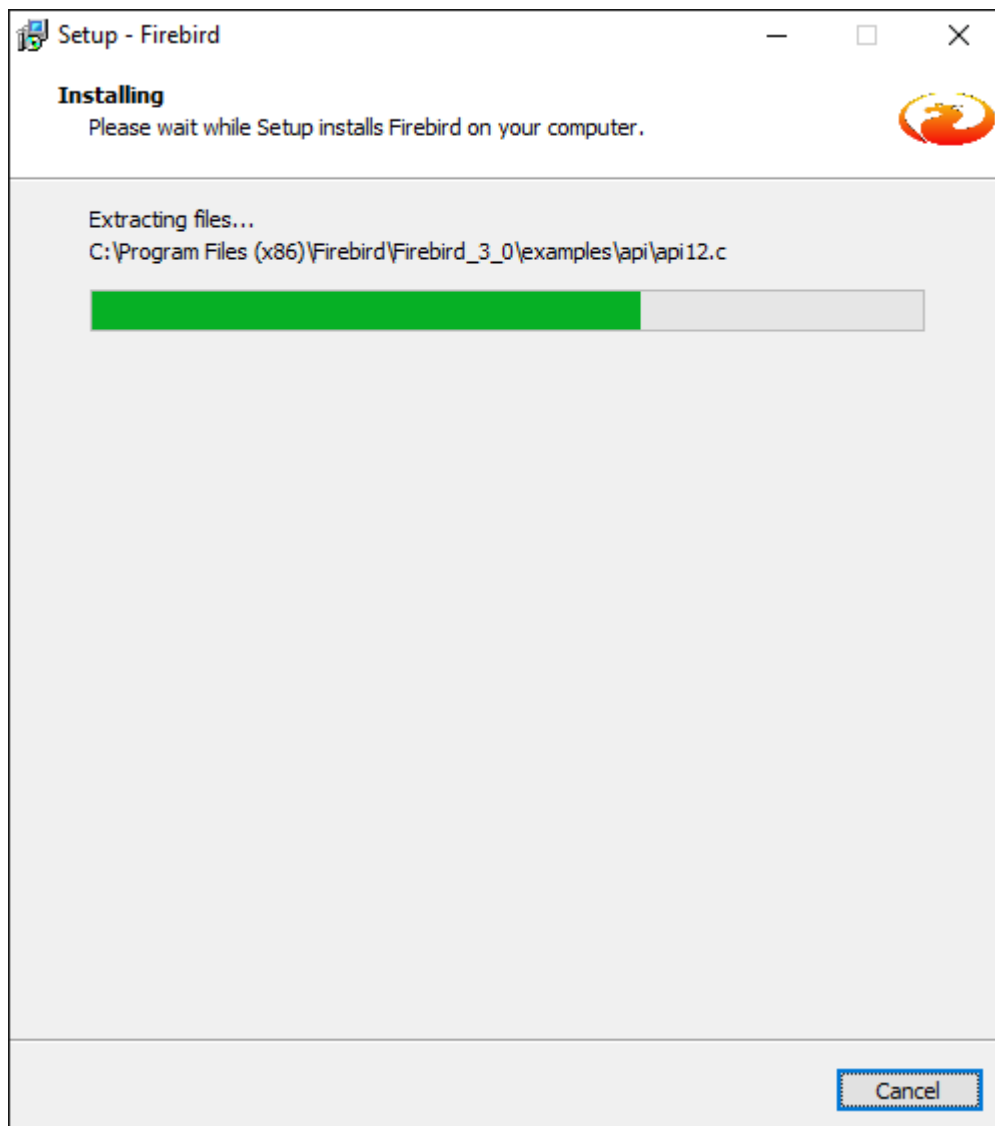


Figura 5.2a - O progresso da instalação

Ao final, o Firebird estará instalado, e a janela da **figura 5.2b** será apresentada uma tela com as informações a respeito da instalação do servidor.

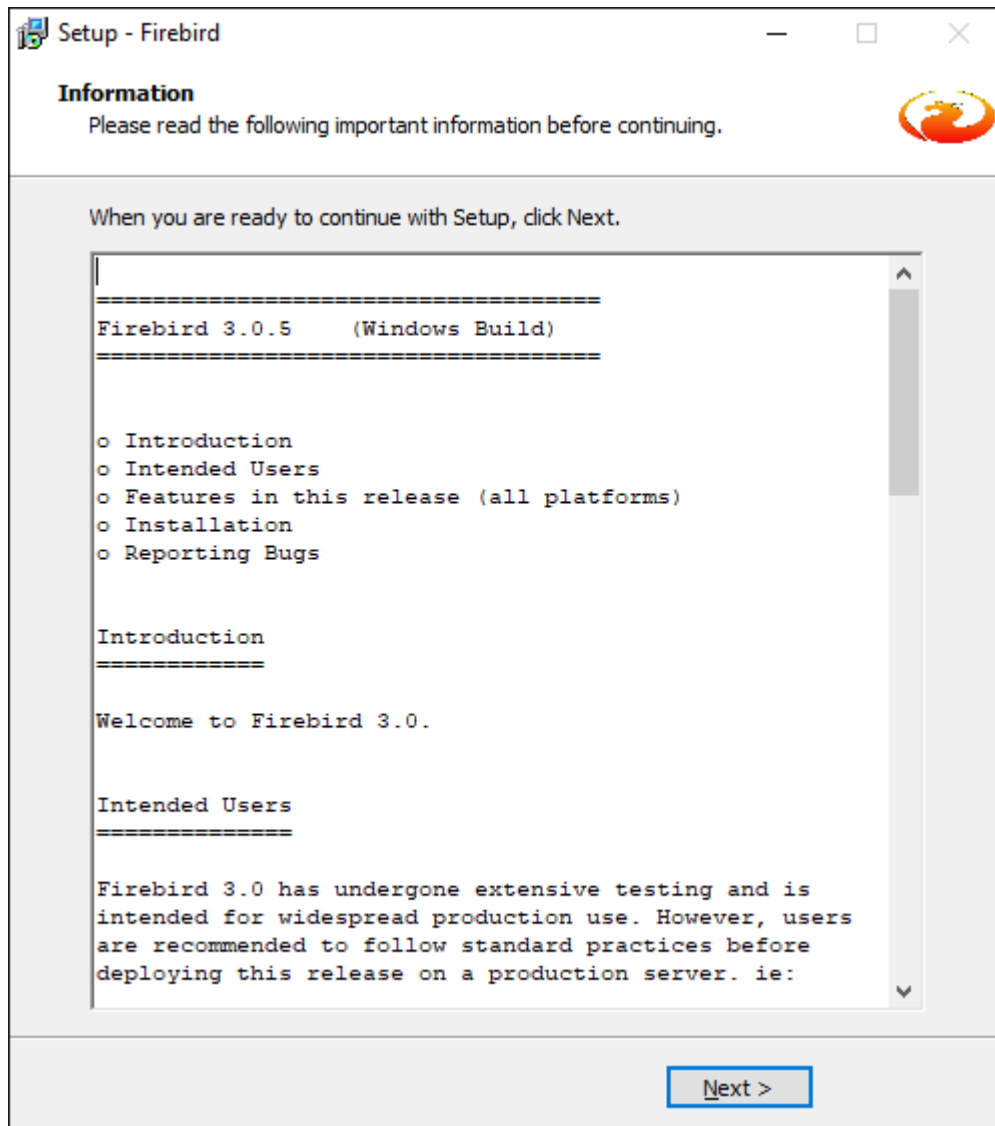


Figura 5.2b - janela de informações sobre a instalação do Firebird

Para finalizar, clique no botão **[Next >]** da **figura 5.2b**. Neste momento será apresentada a janela da **figura 5.2c**, anunciando que o setup instalou com sucesso o Firebird.

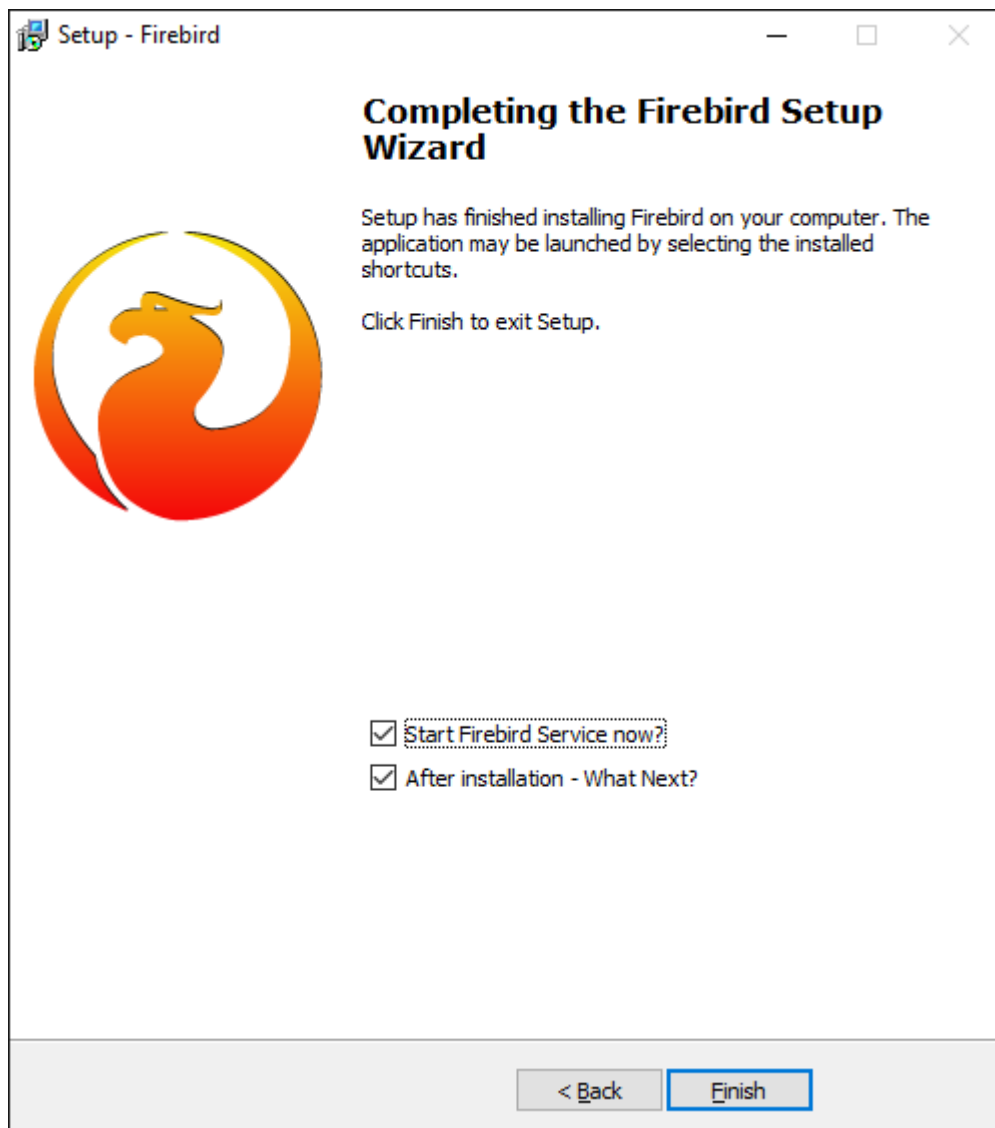


Figura 5.2c - Janela final do Assistente do Setup: servidor pronto para uso

Para finalizar todo o processo, clique no botão **[Finish]** na janela da **figura 52.c**.

Marcando "*Start Firebird Service now?*" o servidor será imediatamente ativado; isto é recomendado para não ter que ativar o servidor manualmente.

A opção "*After installation - Wat Next?*", exibe uma tela com explicações sobre a instalação do Firebird. É interessante essa leitura.

Nota: No link http://www.firebirdnews.org/docs/fb2min_ptbr.html (acesso em 20/05/2020 -16:48) você pode obter mais informações sobre o Firebird.

Nota: Dependendo da versão, pode ser que as opções da **figura 5.h** apareçam um pouco diferentes; entretanto, de um modo geral, é importante atentar para as seguintes informações à respeito da arquitetura a ser escolhida:

- **Usar o Guardian para controlar o servidor?**
Guardian é, segundo Cantu (2006), “*uma aplicação que roda em paralelo com o servidor, monitorando o processo do Firebird*”. Ainda segundo o citado autor, caso alguma coisa derrube o sistema, o Guardian colocará o servido no ar novamente. Por isso é melhor deixar marcada essa opção.
- **Executar como Aplicação ou como Serviço?** Neste caso é fácil decidir; se o sistema está baseado em Windows (NT, 2000, 2003, ou XP) é melhor rodar como Serviço; para outros sistemas operacionais escolha Aplicação. Neste trabalho optaremos para Serviço (processo), pois a plataforma usada será o Windows XP.
- **Iniciar o Firebird automaticamente de cada vez que o sistema arranca (inicia)?**
Deixe marcado para melhor comodidade.
- **Copiar a biblioteca do cliente Firebird para a pasta do sistema?** Esta é uma opção que deve ser marcada, pois assim a biblioteca (fbclient.dll) será automaticamente copiada para a pasta *system32.dll* do Windows, e os programas que a utilizam não precisarão procurá-la em outro local.
- **Criar a biblioteca cliente como GDS32.dll para retro-compatibilidade?** Nas versões anteriores do Firebird, a biblioteca cliente é conhecida como *gds32.dll* (e não como fbclient.dll); assim é melhor deixar marcada essa opção para manter a compatibilidade com versões mais antigas.

A **figura 5.3** mostra o Firebird Tools no desktop do Windows 10; clicando nele, o sistema mostra a tela da **figura 5.4**, com a linha de comando pronta para receber instruções *sql* de criação/manutenção de um banco de dados.

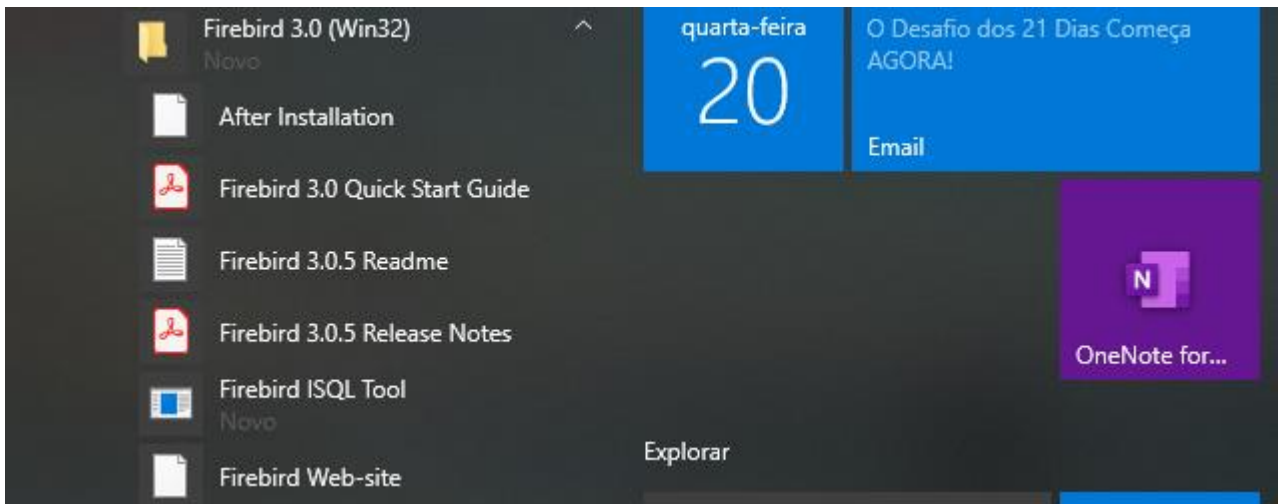


Figura 5.3 - O Firebird 3.0 no desktop do Windows 10 com seus programas

Acessando no Windows 10: *Iniciar/Ferramentas Administrativas* e dando um duplo clique em “**Serviços**”, aparece a janela da **figura 5.4a** com o Firebird na lista dos programas instalados, e em execução.

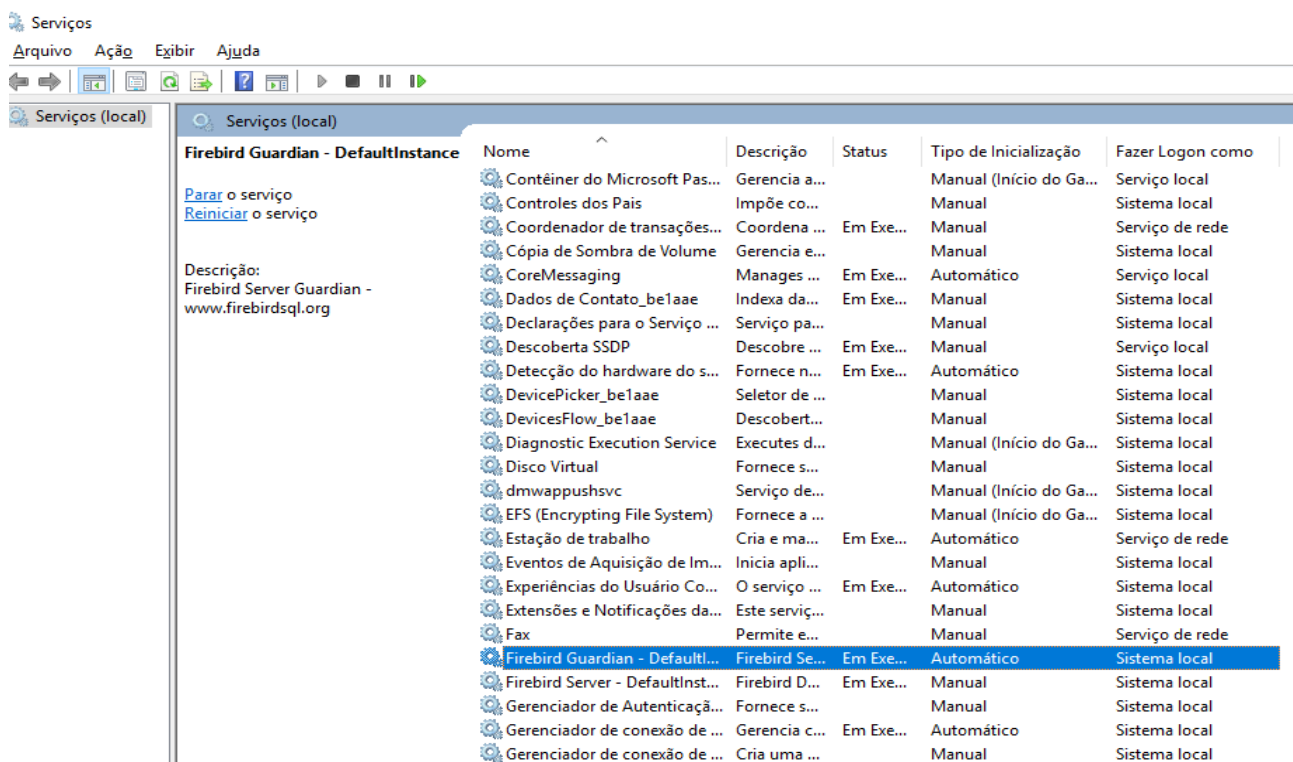


Figura 5.4a - Mostrando o Firebird na lista de programas em Serviço sob o Windows 10

Dano um duplo *click* sobre “*Firebird Guardian - Default*” aparece a janela da **figura 5.4b**, mostrando as condições do Servidor Firebird, sendo executado sob o Windows 10.

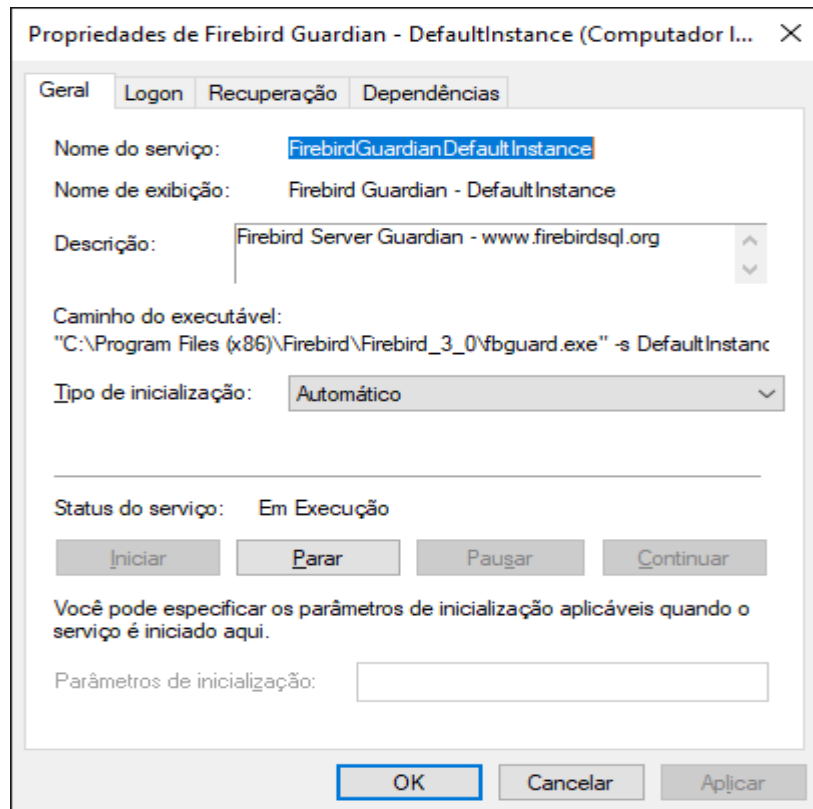


Figura 5.4b - A configura do Servidor Firebird sendo executado sob o Windows 10

5.2.2 - Trabalhando com o servidor Firebird

Depois de instalar o servidor Firebird, o próximo passo é utilizá-lo. Existe o “Firebird ISQL Tools” que pode ser acessado a partir do grupo de programas no *desktop* do Windows 10: **Iniciar/Firebird ISQL Tools**. O resultado dessa ação é uma janela DOS mostrada na **figura 5.5**, onde é solicitada uma ação através de uma linha de comando; a primeira ação, evidentemente deve ser a criação de um banco de dados no servidor, pois todos os outros elementos dependem de se ter um banco criado.

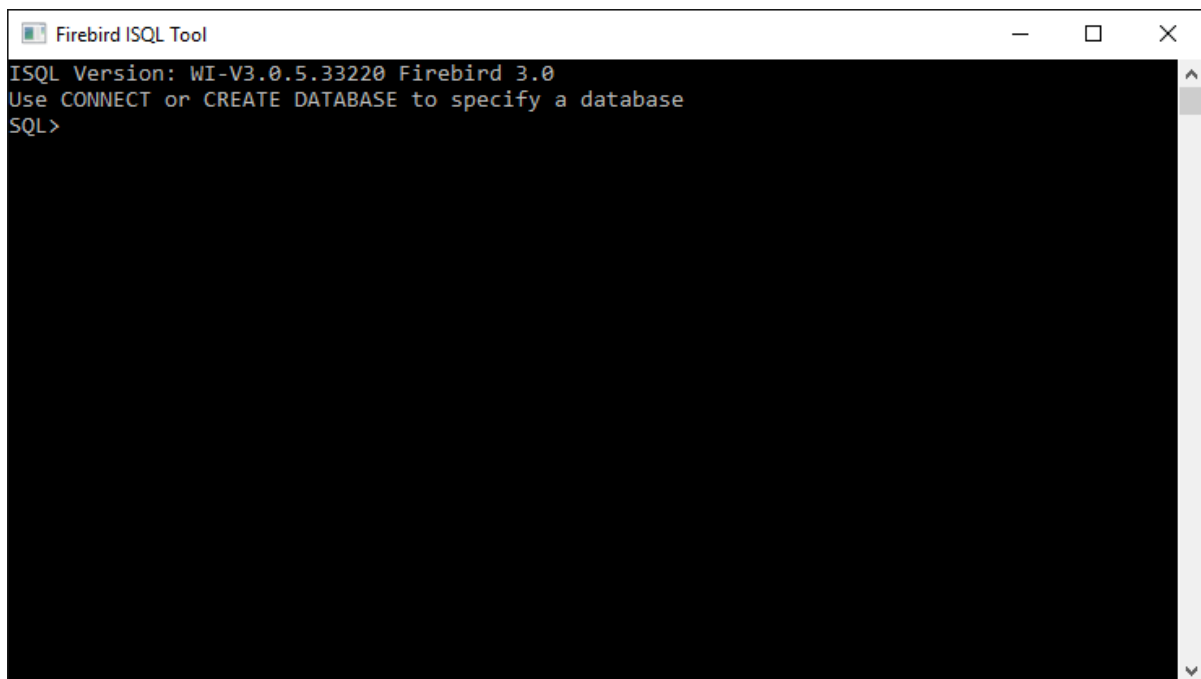


Figura 5.5 - Janela DOS para manipulação de banco de dados no servidor Firebird

Executar linhas de instruções através de comandos no *prompt* do DOS é por demais cansativo e improdutivo quando se deseja trabalhar com um banco de dados de maneira mais ágil. Para resolver este pequeno problema é que existem interfaces gráficas que permitem uma interação mais direta entre o usuário e o banco de dados. Uma dessas ferramentas é o IBExpert, tratado no item seguinte.

5.3 - A Ferramenta IBExpert

O IBExpert, criado e distribuído pela empresa “HK Software”, pode ser baixado pela Internet, sob determinadas condições, e é preciso adquirir uma licença para seu uso contínuo. Versões *trial* podem ser baixadas no *site* oficial da empresa: www.ibexpert.com/. Comparado com a tela da **figura 5.5** (linda de comandos) o IBExpert oferece muitas vantagens, como, por exemplo, a criação/manutenção de objetos do banco de dados de modo totalmente interativo sem necessidade de digitar instruções sql manualmente. A tela da **figura 5.6**^[14] mostra uma tela que se apresenta para fazer o download da ferramenta. [

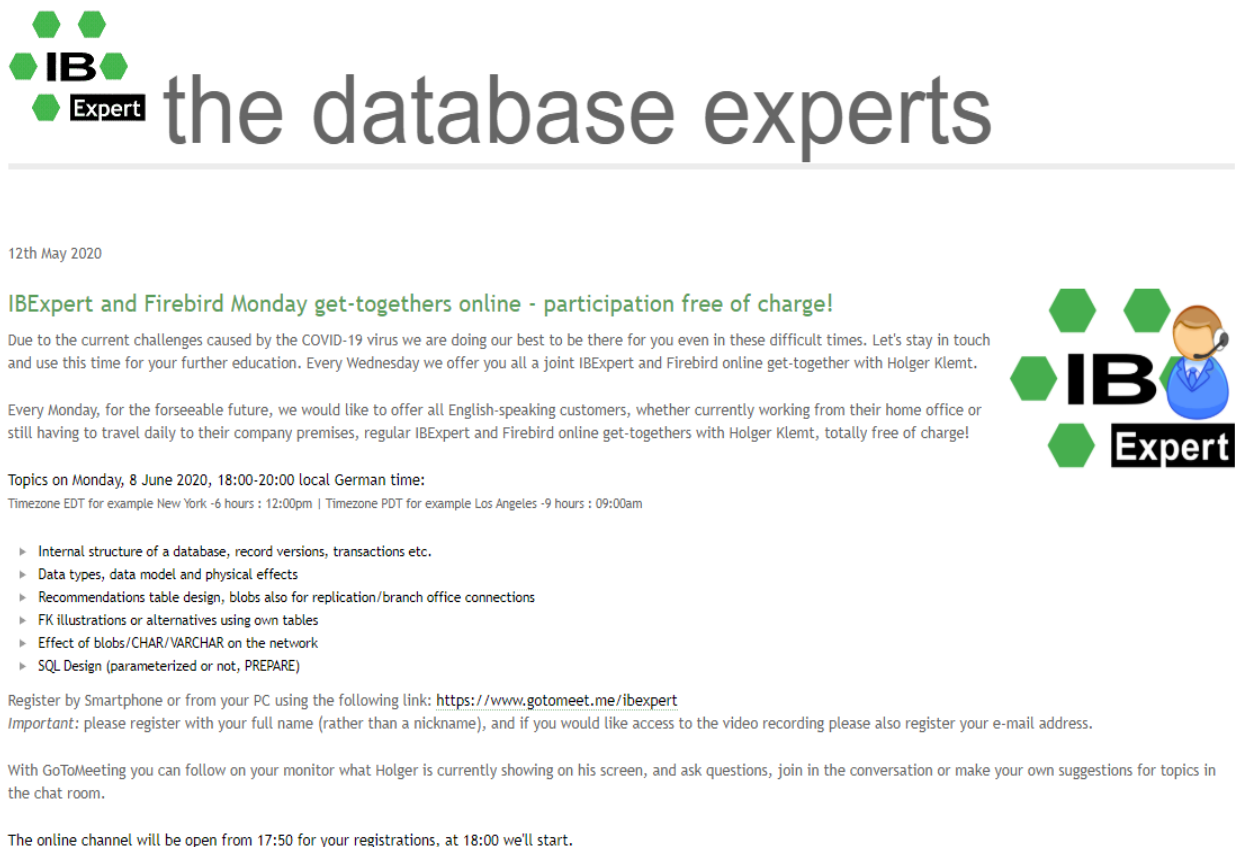


Figura 5.6 - Tela de download do IBExpert

Depois de instalada a ferramenta deve ser carregada; a primeira tela a se apresentar é a janela principal onde o primeiro elemento a ser criado deve ser o banco de dados. A **figura 5.7** mostra essa janela.

¹⁴ A tela da **figura 5.6** pode se apresentar diferente, dependendo da versão do **IBExpert**; mas, de qualquer forma, é preciso fazer o *download* e instalar essa ferramenta para gerenciar bancos de dados; o Firebird é um deles.

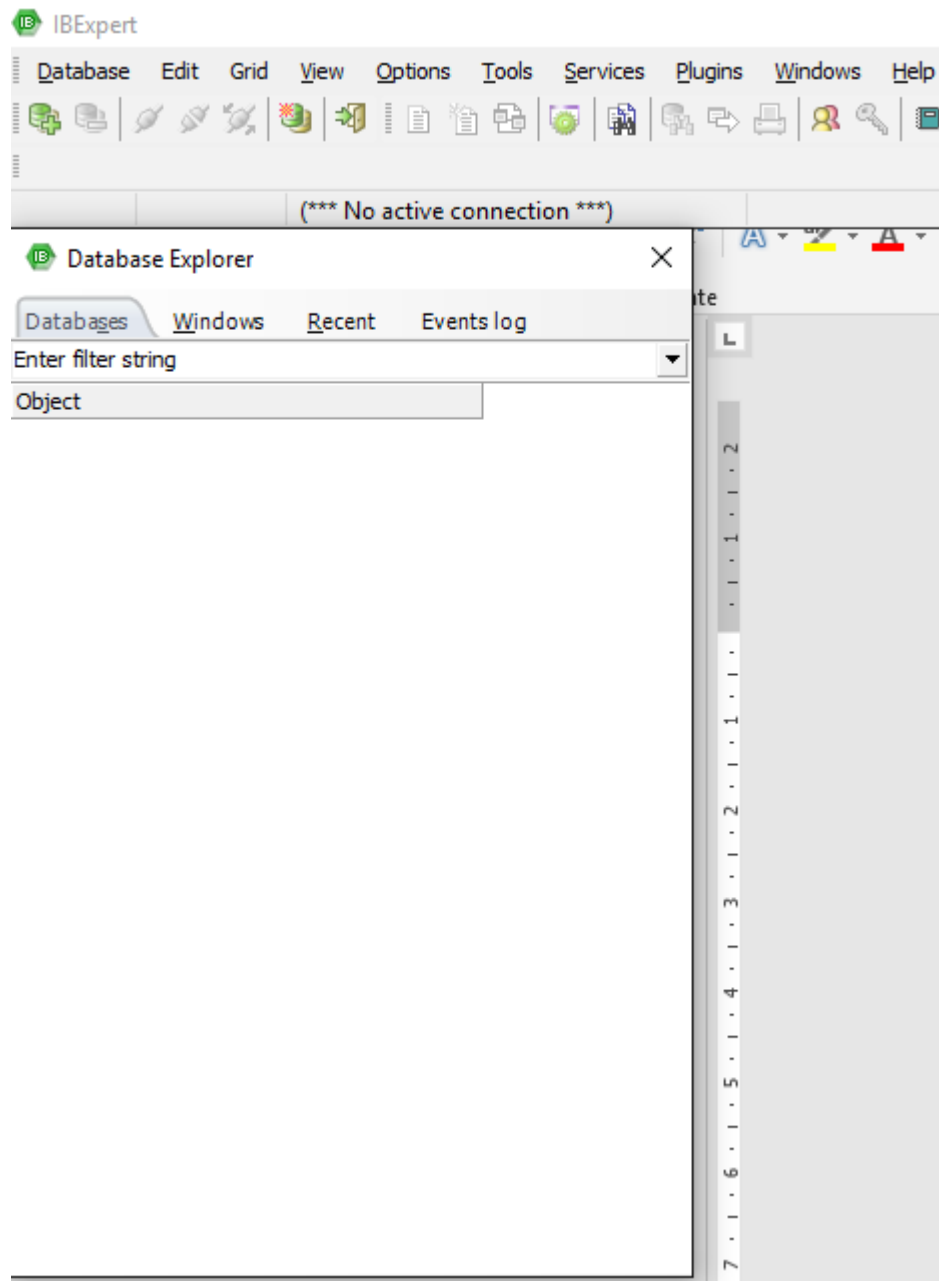


Figura 5.7 - Janela do IBExpert com menu principal

Para ilustrar a criação de um banco de dados no servidor Firebird, vamos considerar um banco de dados denominado “Locadora”, baseado no MER da **Figura 5.10**, com as seguintes tabelas:

- ✓ **Clientes** Dados pessoais dos clientes.
- ✓ **Genero** Dados sobre o gênero do filme (ação, ficção, terror, guerra, etc).
- ✓ **TipoFilme** Dados sobre o tipo de filme (lançamento, catálogo, etc).
- ✓ **Diretor** Dados sobre o diretor do filme.
- ✓ **Filmes** Dados sobre o filme (código, títulos, gênero, tipo, etc).
- ✓ **Locacoes** Dados sobre locação (cliente, datas, filme, etc).
- ✓ **Atores** Dados sobre os atores dos filmes (código, nome e nacionalidade).
- ✓ **AtorFilme** Dados sobre o ator/filme(código do ator, filme e tipo de participação).
- ✓ **Participacao** Dados sobre o tipo de participação do ator no filme (código e descrição).

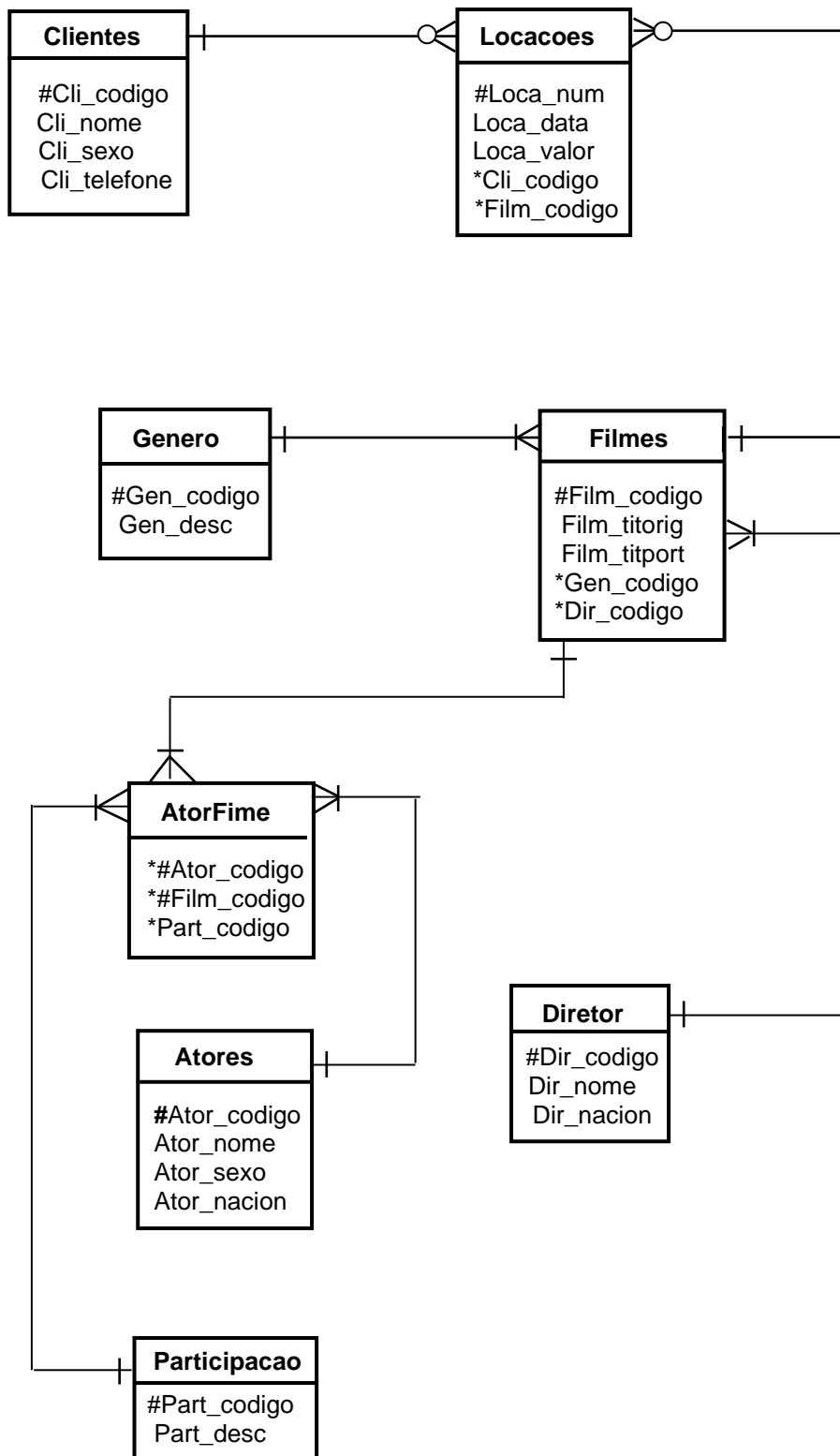


Figura 5.10 - MER do banco "Locadora"

Nota: No MER da Figura 5.10 estamos considerando que cada filme locado representa apenas uma locação individual; assim o relacionamento cliente-locação tem cardinalidade 1:N e não N:M como se poderia pensar.

5.3.1 - Criando um banco de dados no servidor local

Para criar (gerencial) um banco com o IBExpert clique em **DataBase** e selecione a opção “**Cretae Database...**”. A janela da **figura 5.8a** se apresenta para dar início ao processo.

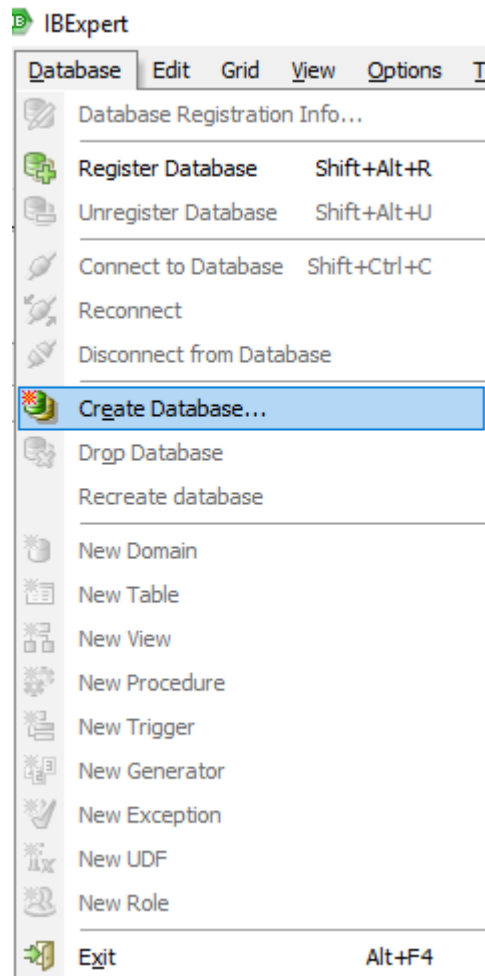


Figura 5.8a - Criando um banco com o IBExpert: passo 1

1 - Clique no *menu Database/Create Database...*; a janela que se apresenta é a da **figura 5.8b** com alguns campos para serem preenchidos.

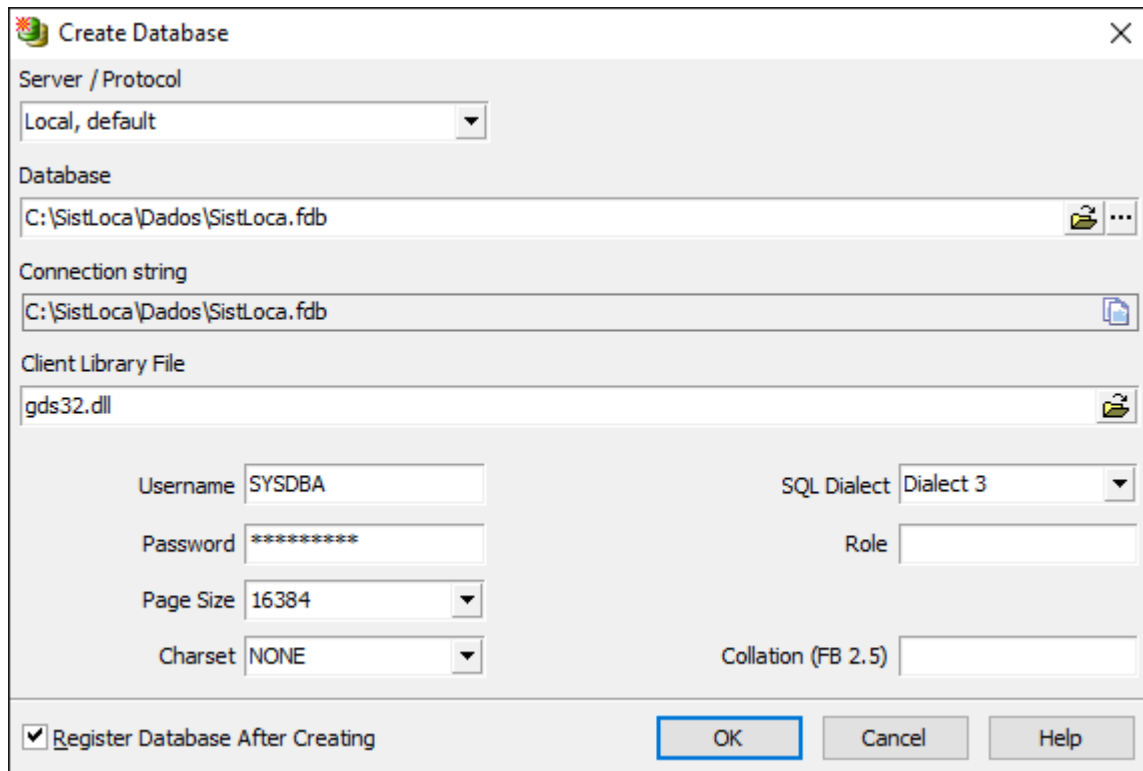


Figura 5.8.b – Criando um banco no Servidor Firebird

2 - Preencha os campos relativos ao novo banco de dados do seguinte modo:

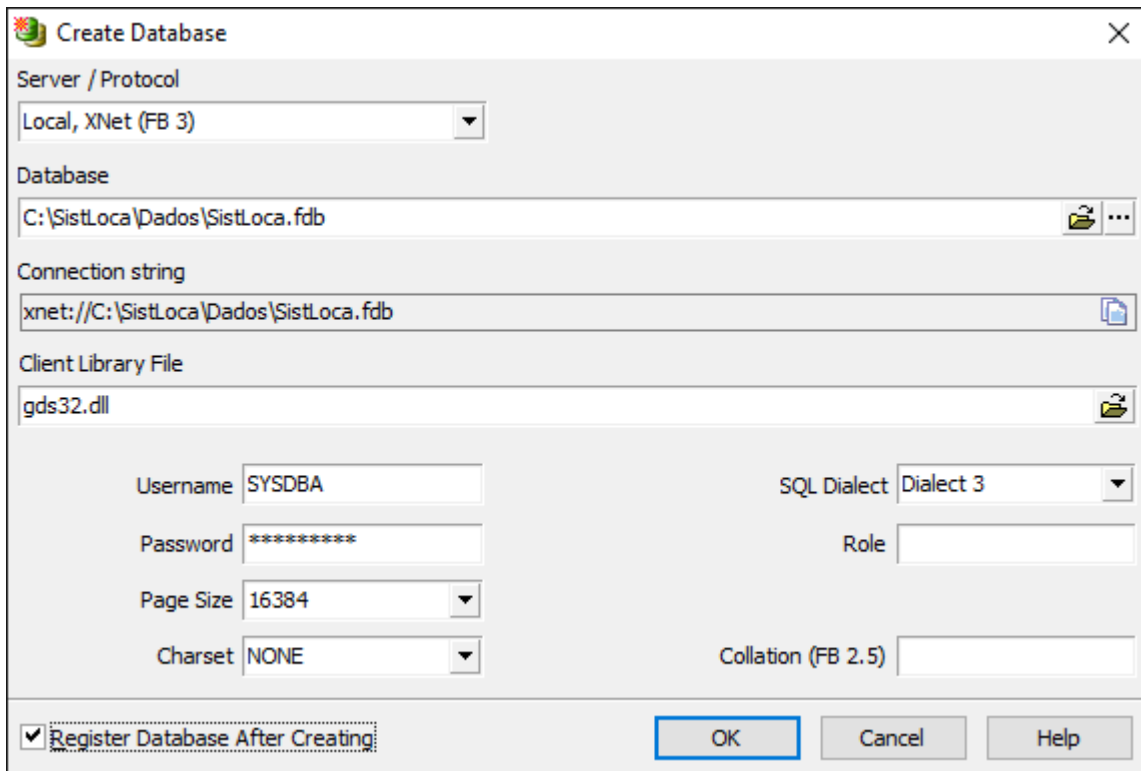
- Server **Local** (selecionado na caixa *combo*).
- Database Nome do banco de dados com o caminho completo ^[15].
- Username **SYSDBA**
- Password **masterkey** (tudo em minúsculas).
- Page Size Tamanho da página (16384).
- Charset (NONE, algum de sua conveniência).
- SQL Dialect **3** (o mais indicado atualmente).
- Register Database After Creating: Deixe marcado.

Preenchendo os campos requeridos e confirmando no botão **[OK]** o resultado é a **Figura 5.12**.

Nota: Às vezes, na hora de registrar ou conectar o banco pode ser que aconteça aparecer uma mensagem dizendo “*que o banco não está utilizável*”. Este aviso pode indicar que, ou o banco não foi encontrado no caminho indicado em “Database” da **Figura 5.8.b**, ou qualquer outro problema na conexão. Este caso pode ser solucionado colocando “localhost” na frente do *drive* do caminho onde está o banco; por exemplo: **localhost:D:\CursoDB\Locadora.fdb**

Atenção: Na propriedade *DatabaseName* do componente **TIDatabase** isto também deve ser feito

¹⁵ É possível usar qualquer extensão para um banco de dados Firebird; entretanto, o padrão adotado pelo IBExpert é .FDB (Firebird Data Base).



The image shows a 'Create Database' dialog box with the following fields and options:

- Server / Protocol:** Local, XNet (FB 3)
- Database:** C:\SistLoca\Dados\SistLoca.fdb
- Connection string:** xnet://C:\SistLoca\Dados\SistLoca.fdb
- Client Library File:** gds32.dll
- Username:** SYSDBA
- Password:** *****
- Page Size:** 16384
- Charset:** NONE
- SQL Dialect:** Dialect 3
- Role:** (empty field)
- Collation (FB 2.5):** (empty field)
- ☒ **Register Database After Creating**
- Buttons:** OK, Cancel, Help

4 - Confirme os dados do banco clicando no botão **[OK]**. Em seguida será apresentada a tela da **Figura 5.13** onde deve ser definido o *alias* do banco e a versão em que será criado. Preencha esses dados e confirme clicando no botão **[Register]** para que o banco seja registrado no servidor. Assim, após ser devidamente registrado, o banco será apresentado no “Database Explorer” com o seu *alias*. Observe a **Figura 5.14** como fica.

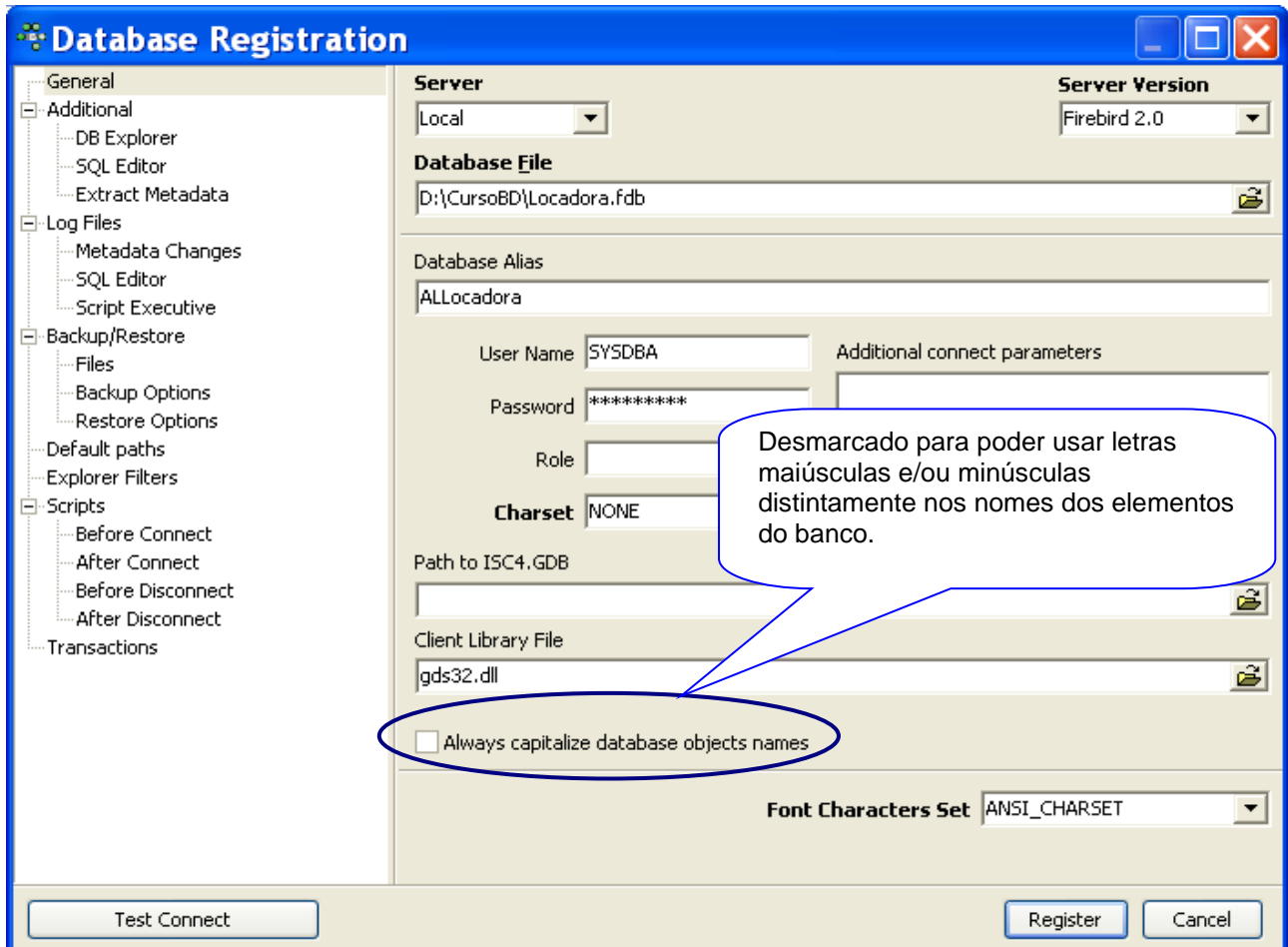


Figura 5.13 - Definindo o *alias* para o banco e a versão do Firebird

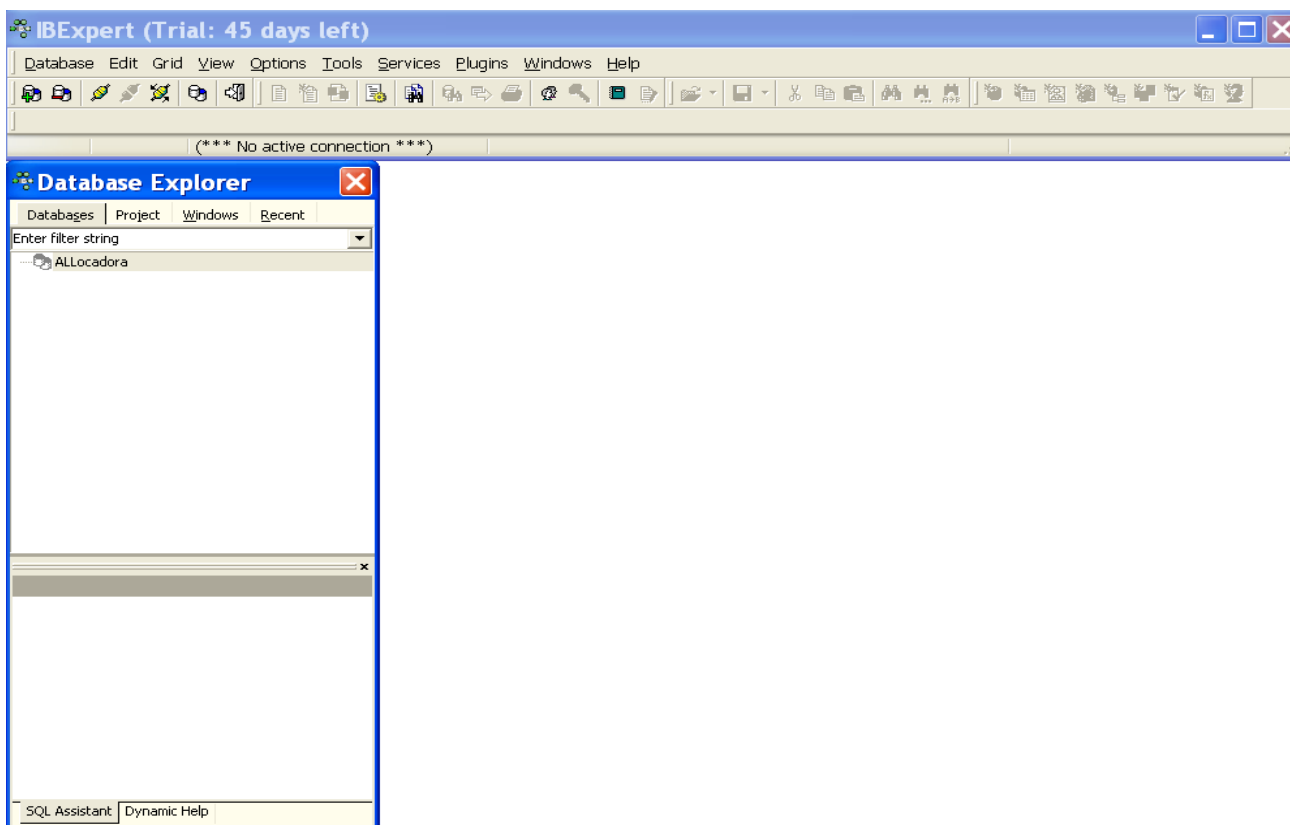


Figura 5.14 - Exibindo o banco de dados no "Database Explorer"

Atenção: Caso não tenha optado por registrar o banco no servidor por ocasião de sua criação, isto tem que ser feito posteriormente no *menu Database/Register Database* e preenchendo os dados requeridos nos campos indicados:

- Server: **Local;**
- Server Version: **Firebird 2.0**
- Database File: Nome do banco de dados com o caminho completo.
- Database Alias: Apelido (*alias*) do banco de dados.
- User Name: **SYSDBA**
- Password: **masterkey** (tudo em minúsculo) .
- Role: (vazio) .
- CharSet: **NONE;**
- Path to ISC4.GDB (vazio)^[16]
- Cliente library File: **gds32.dll**^[17] (o próprio sistema já a considera) .
- Font Characters Set **ANSI_CHARSET**

Nota

Os tipos de dados suportados pelo Firebird são os mesmos do Interbase 6.x com alguns outros tipos adicionais. Entretanto, tal como no Interbase, o Firebird ainda não contempla o tipo BOOLEAN (lógico). Mas isto pode ser contornado, criando um domínio (*domain*) assim:

```
CREATE DOMAIN Logico AS CHAR(1) CHECK (VALUE IN ('S', 'N')) NOT NULL;
```

Desse modo será criado o domínio denominado Logico (sem acento) cujos valores devem ser apenas um dos dois: **'S'** (Sim) ou **'N'** (Não), e obrigatório.

Depois de criado e devidamente registrado, o banco de dados estará pronto para ser usado; mas antes, uma última etapa deve ser cumprida: a sua conexão. Para conectar o banco ao servidor, clique com o botão direito sobre o *alias* e em seguida selecione a opção "Connect to Database" ou simplesmente dê um duplo clique sobre o *alias*. O resultado é mostrado na **Figura 5.15**, onde o banco de dados aparece como conectado e com todos os objetos que poderão ser criados e gerenciados: tabelas, visões, procedimentos, *triggers*, *generators*, *exceptions*, UDFs, roles (privilégios), índices e *scripts*.

¹⁶ O ISC4.GDB é um arquivo ligado à segurança de bancos de dados Interbase/Firebird. Você pode indicar o caminho, mas normalmente ele é gravado em: **C:\Arquivos de Programas\Borland\Interbase**

¹⁷ Biblioteca que faz a interface entre a aplicação e o banco de dados Interbase/Firebird.

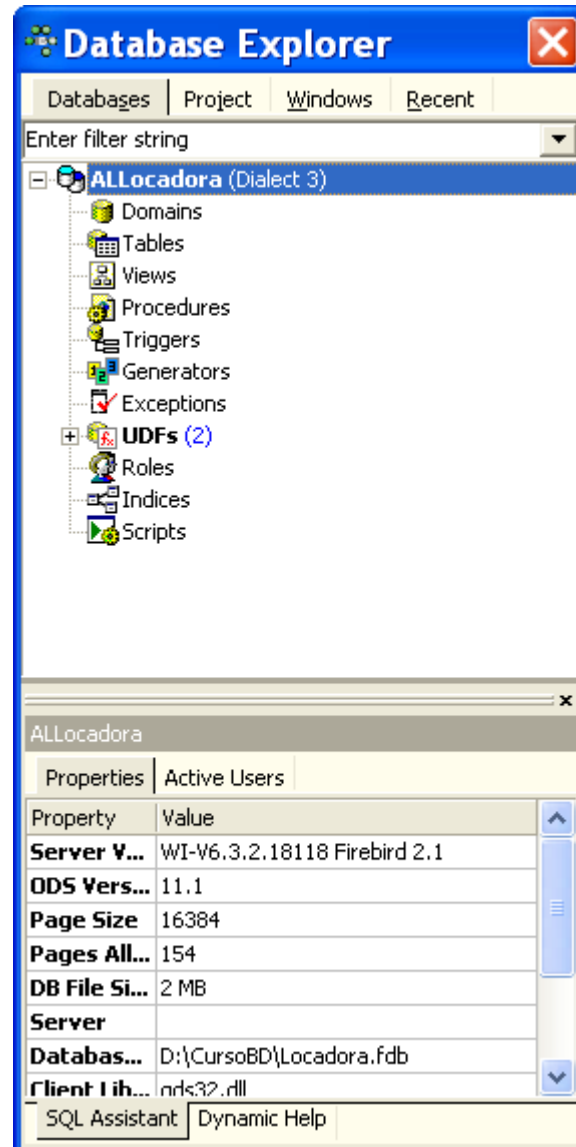



Figura 5.15 - O banco “Locadora” conectado ao servidor Firebird

5.4 - Criação de tabelas com o IBExpert

Para trabalhar com o IBExpert é necessário, primeiramente, criar as tabelas, pois são elas os únicos objetos que armazenam dados reais. E, diferentemente do IBConsole (outra ferramenta de gerenciamento de banco de dados), a criação de tabelas no IBExpert é visual, tal como no Access; isto permite uma maior interação com o usuário, já que não há necessidade de um conhecimento maior da linguagem SQL. Os procedimentos para a criação de tabelas são os apresentados a seguir...

1 - Selecione o menu **Database/New Table** ou então clique com o botão direito do mouse sobre o ícone  **Tables** dentro do banco ALLocadora, e em seguida selecione a opção “New Table”. A janela da **Figura 5.16** aparece para que seja definida a estrutura da nova tabela.

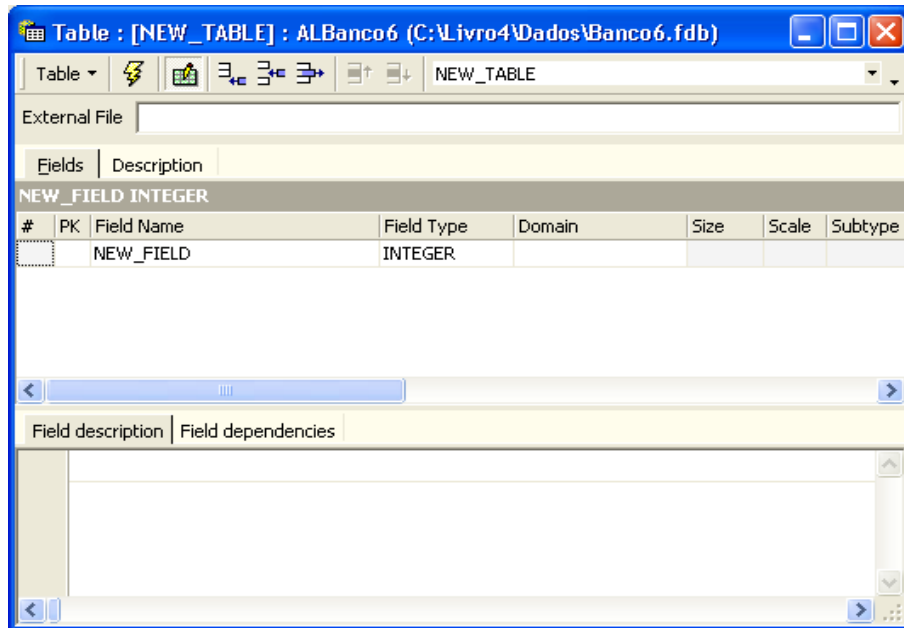




Figura 5.16 - Criando uma tabela com o IBExpert: 1º Passo

2 - Preencha os campos requeridos para criar a tabela “Clientes”. Veja como fica na tela da **Figura 5.17**. Nessa figura pode ser notado como o IBExpert é interativo com o usuário; o processo de definição da estrutura da tabela é auto-explicativo. É possível, até, entrar com os domínios e os *checks* para validar os campos. O ícone da chavinha  indica que esse campo é uma chave primária. Se outro campo também fizesse parte da chave primária, um segundo ícone (agora com índice 2) seria colocado ao lado desse outro campo. Para definir/eliminar uma chave, basta dar um duplo clique sobre o quadradinho antes do campo.

Para criar com um campo basta clicar no ícone de adicionar novo campo  e continuar. Note que podemos também decidir se o campo será ou não obrigatório, marcando ou desmarcando a caixinha de verificação da opção **Not Null**; no caso o código e o nome do cliente não podem ficar em branco, e por isto foram marcados como Not Null. Outra coisa a ser notada na **Figura 5.17** é que podemos digitar um texto explicativo para cada campo; no caso está sendo explicado (obviamente) que o campo “Cli_codigo” é a chave primária da tabela. Quando existem muitos campos com dados parecidos essa ação pode ser muito importante para esclarecer o usuário.

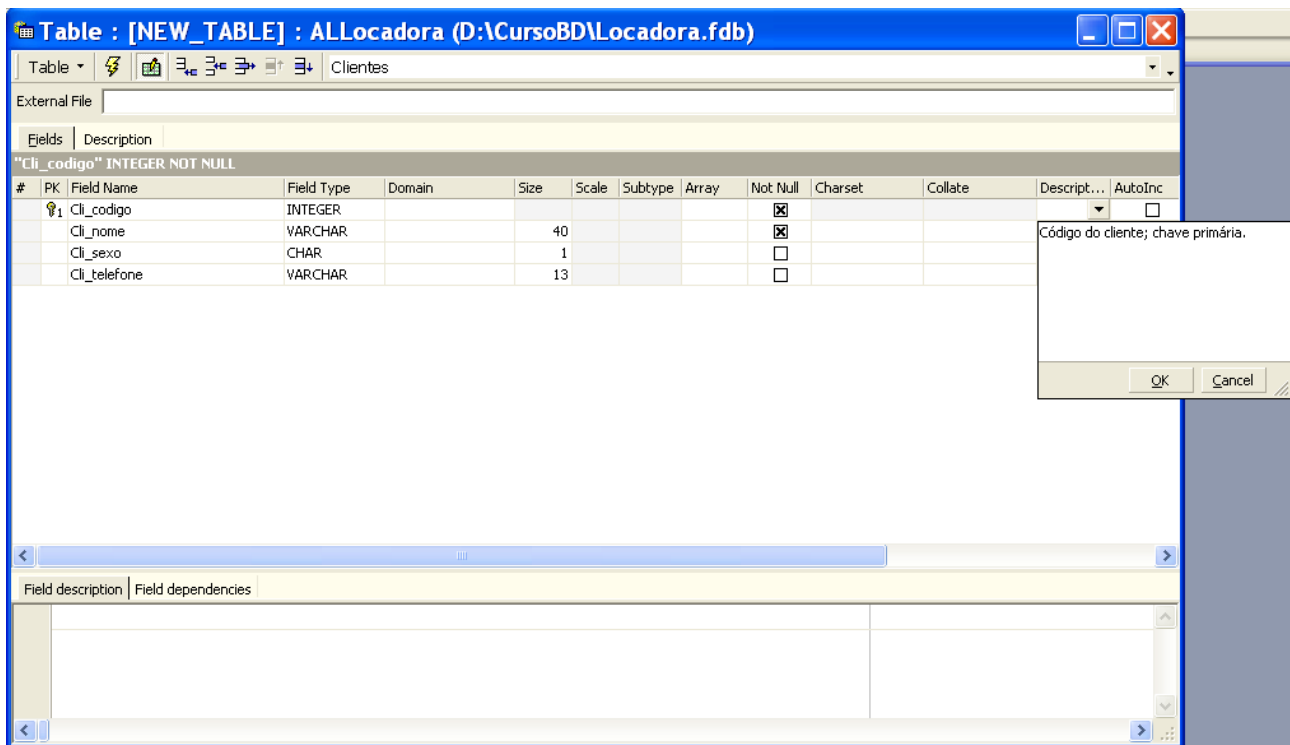


Figura 5.17 - Criando tabelas com o IBExpert: 2º Passo

3 - Compile a criação da tabela no **menu Table/Compile**, ou dê um clique sobre o ícone . O resultado é a janela da Figura 5.18.

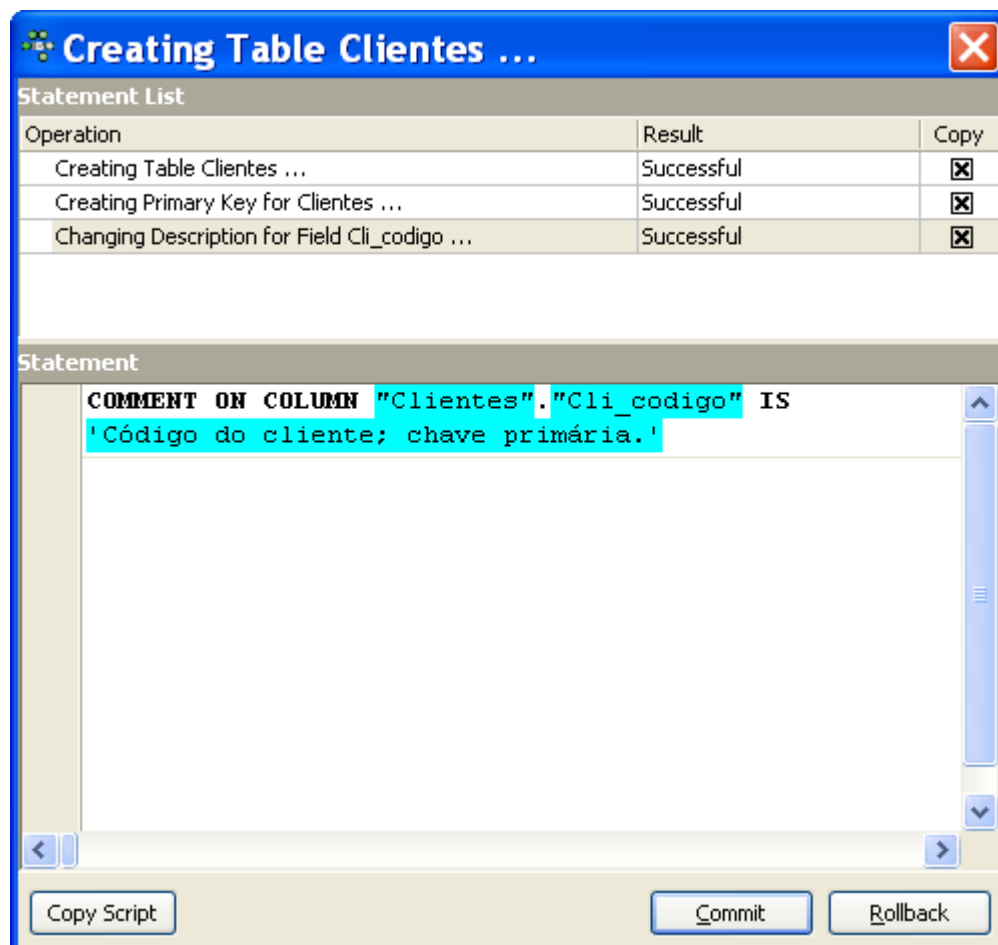


Figura 5.18 - Criando tabelas com o IBExpert: 3º Passo

4 - Confirme no botão **[Commit]**. Se tudo estiver correto você ouvirá um alerta sonoro indicando que a criação da tabela teve sucesso e o processo reexibirá a janela de criação da tabela, agora com seu nome definitivo (veja na **Figura 5.19**).

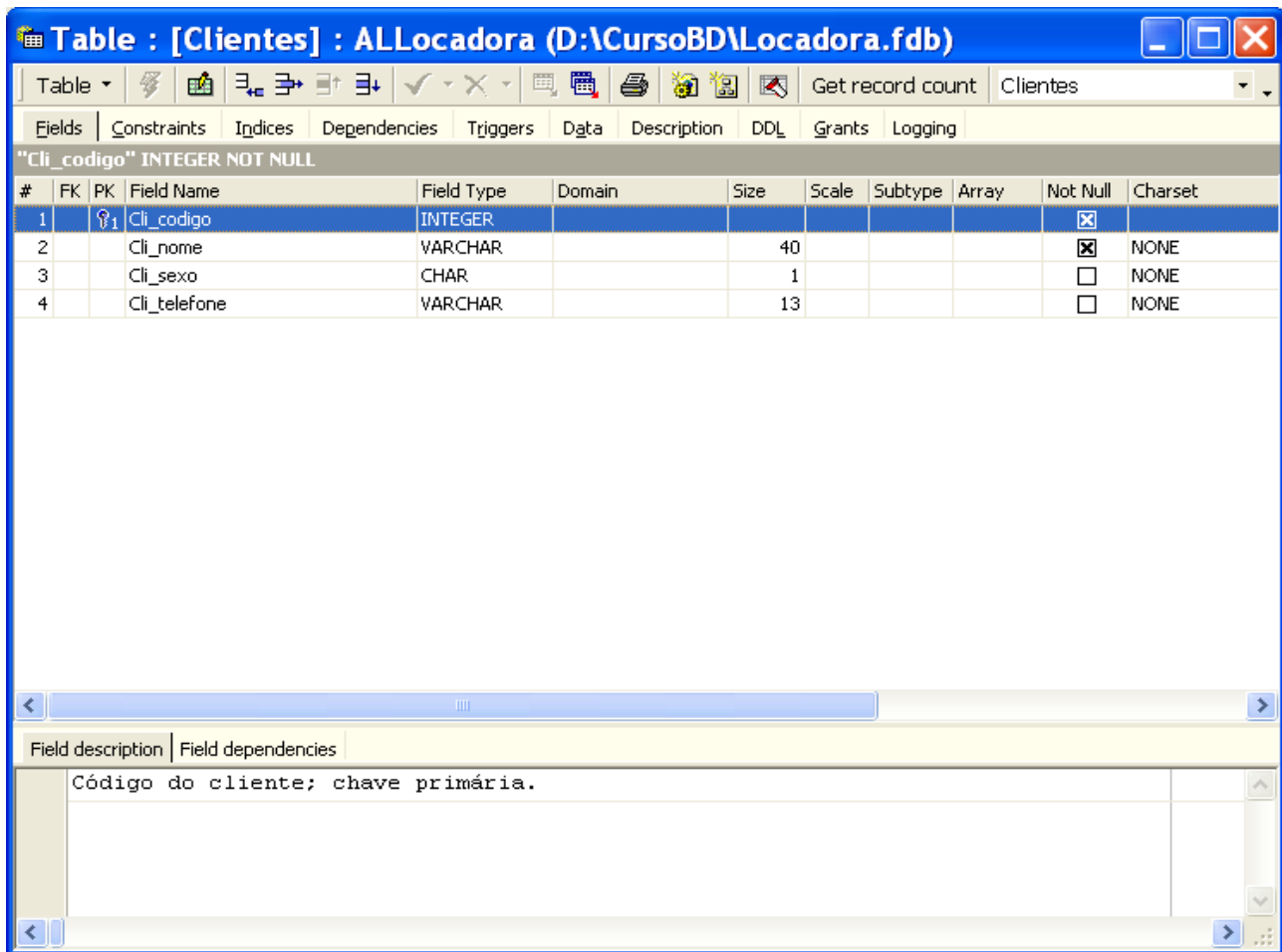


Figura 5.19 - Criando tabelas com o IBExpert: 4º Passo

Conforme foi dito antes, além de tabelas o IBExpert gerencia muitos outros objetos de um banco de dados; confira na **Figura 5.20**, onde está evidenciada a nova tabela “Clientes”.

- Views:** Visões diferentes dos registros da tabela (resultados de consultas).
- Procedures:** Rotinas armazenadas no servidor.
- Triggers:** Gatilhos de validação de entrada de dados.
- Generators:** Instruções para gerar campos auto-incrementáveis.
- Exceptions:** Exceções que podem ser utilizadas juntamente com *triggers*.
- UDF's:** Funções definidas pelo usuário.
- Roles:** Regras de privilégios para usuários do banco.
- Índices:** Índices (chaves secundárias) criados nas tabelas.

Agora basta repetir os mesmos passos da criação da tabela “Clientes” para criar as outras tabelas do banco “Locadora”. Mas, é importante frisar que a tabela que possui alguma chave estrangeira só poderá ser criada depois de sua tabela-mãe. E assim, criada uma tabela, os mesmos passos devem ser repetidos para criar todas as outras tabelas; E para isto clique com o botão do *mouse* sobre o ícone **Tables** e selecione “New Table...”; em seguida execute os mesmos procedimentos feitos para criar a tabela “Clientes”. A **Figura 5.21** mostra o nosso banco com todas as suas tabelas criadas, incluindo suas chaves (primárias e estrangeiras), em destaque a tabela “AtorFilme”. E sobre essa tabela é importante notar seus campos: “Ator_codigo”, “Filme_codigo” e “Part_codigo”.

Ator_codigo - Código do ator.

Filme_codigo - Código do filme em que o ator atua.

Esses dois campos juntos formam a chave primária da tabela “AtorFilme”

Part_codigo - Código do tipo de participação do ator no filme.

Observe, como foi destacado acima, que a tabela “AtorFilme” possui chave primária dupla: “Ator_codigo+Filme_codigo”. Isto quer dizer que para localizar um registro dessa tabela é necessário o código do filme além do código do ator, pois um ator pode atuar em vários filmes, assim como um filme pode ter vários atores atuando. Na verdade, a tabela “AtorFilme” é uma tabela Associativa oriunda do relacionamento entre as tabelas “Atores” e “Filmes”. O MER da **Figura 5.10** indica esse tipo de relacionamento; as duas primeiras chaves estrangeiras formam a chave primária da tabela.

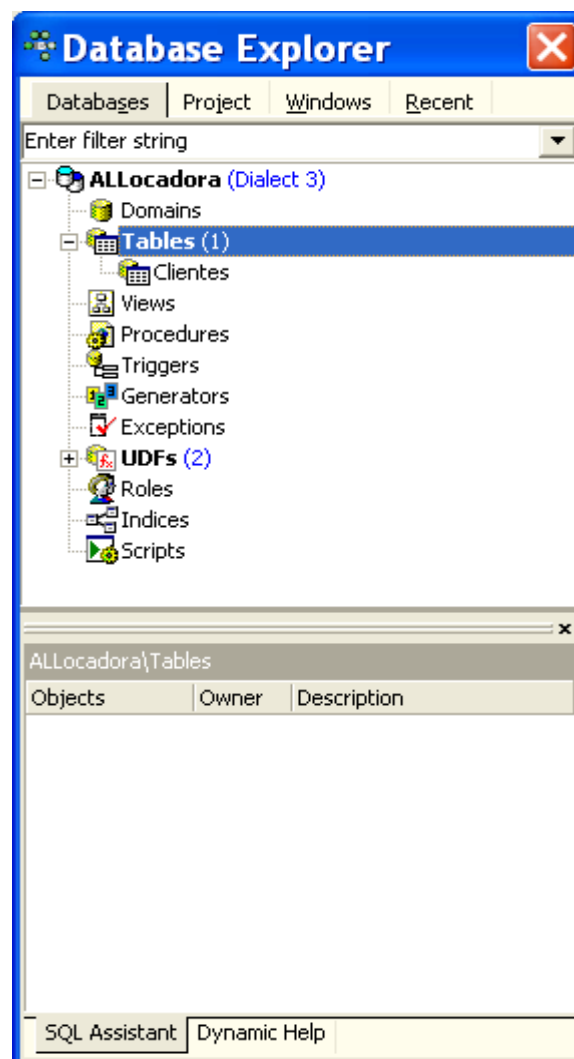


Figura 5.20 - Tabela “Clientes” em destaque no Data Explorer

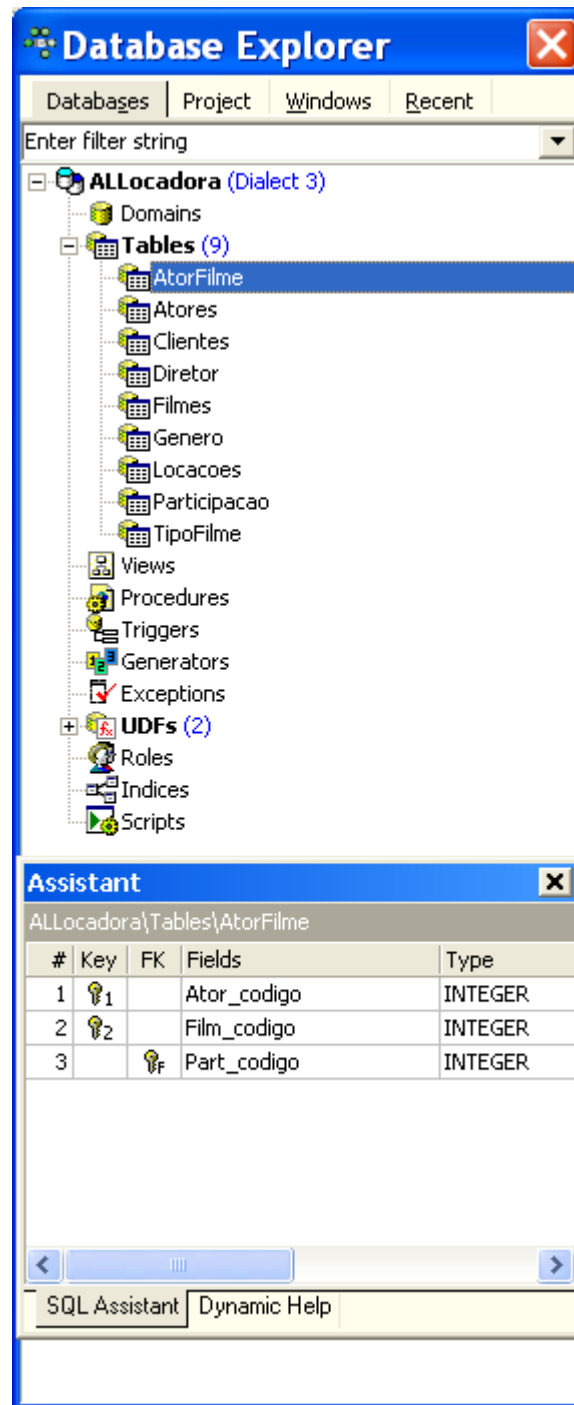


Figura 5.21- O banco “Locadora” com suas tabelas

5.5 - Criação de chaves estrangeiras com o IBExpert

Conforme mostra a **Figura 5.21**, as tabelas possuem suas chaves: primária e estrangeiras. A criação de chaves estrangeiras em tabelas, empregando o IBExpert, é muito fácil; o processo é totalmente interativo, sem a necessidade de se usar instruções sql explícitas. Para ilustrar, vamos mostrar como foram criadas as chaves estrangeiras da tabela “Filmes”. Conforme indica o MER da **Figura 5.10**, essa tabela depende de outras três tabelas: “Genero”, “TipoFilme” e “Diretor”; isto quer dizer que antes de criá-la é necessário criar essas outras três tabelas.

1º Passo - Dê um duplo clique sobre o nome da tabela “Filmes” na relação das tabelas do banco, na janela do “Database Explorer”. Em seguida o resultado é a exibição da estrutura da tabela, conforme mostra a **Figura 5.22**.

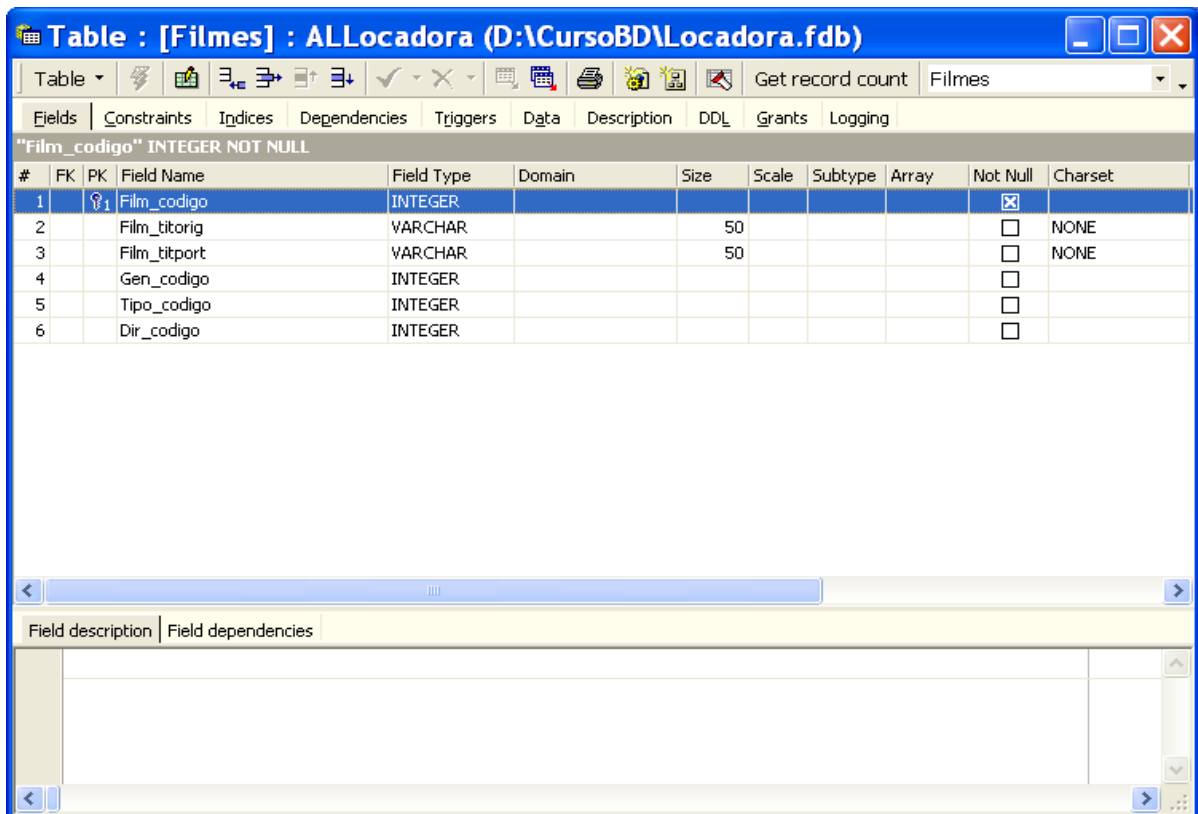


Figura 5.22 - Criando chave estrangeira na tabela “Filmes”: 1º Passo

2º Passo - Selecione o campo “Gen_codigo” e clique na guia “Constraints”; em seguida selecione “2.Foreign keys”; veja como fica na **Figura 5.23** após executar estes dois procedimentos.

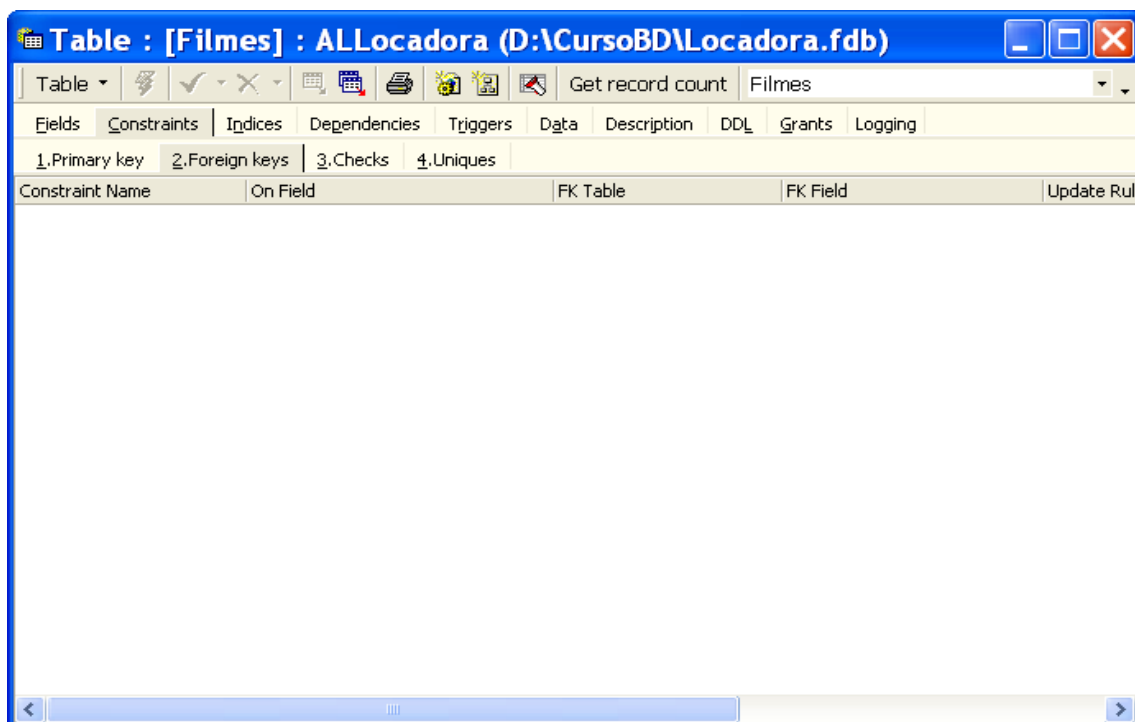


Figura 5.23 - Criando chave estrangeira na tabela “Filmes”: 2º Passo

3º Passo - Clique no centro da janela (parte branca) com o botão direito do *mouse* e escolha opção “New foreign key Ins”; o resultado é mostrado na **Figura 5.24**.

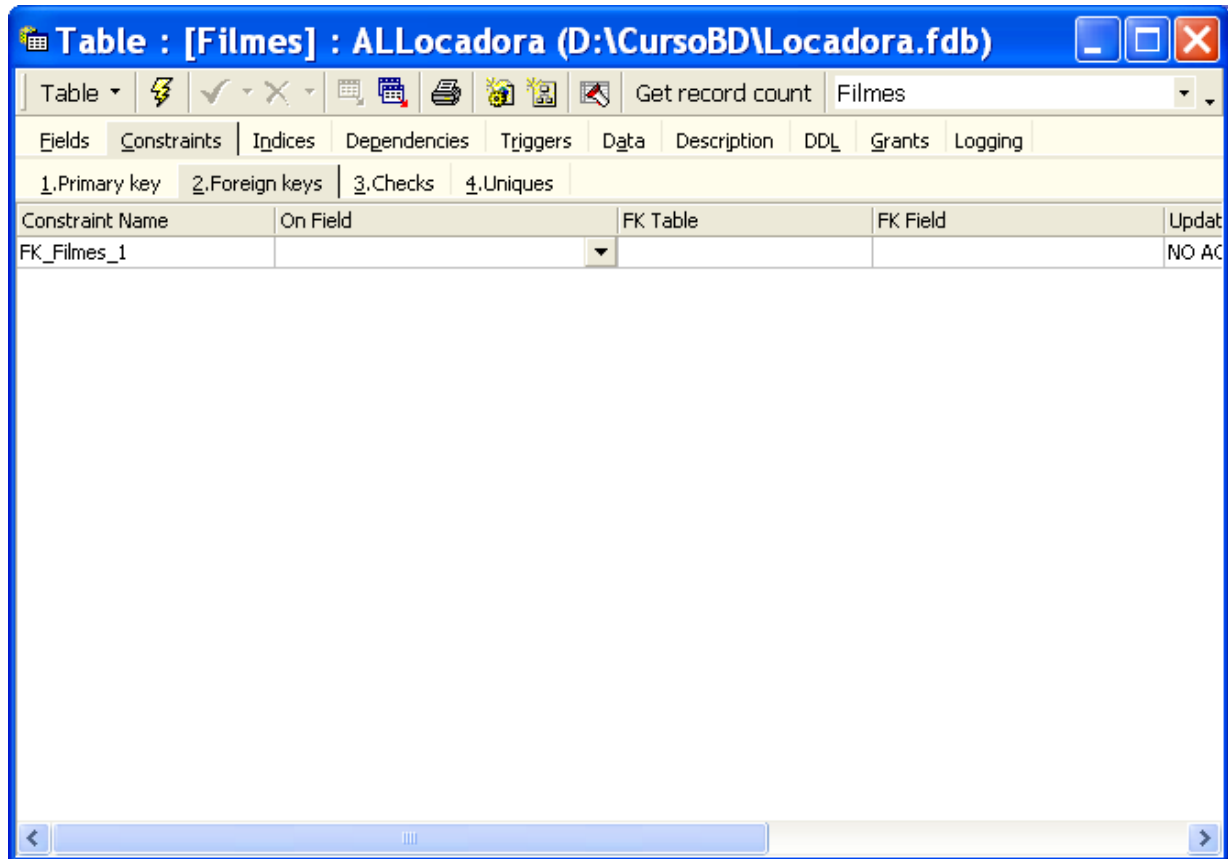


Figura 5.24 - Criando chave estrangeira na tabela “Filmes”: 3º Passo

4 - Defina os campos conforme solicitado.

- **Constraint Name** Nome do índice restritivo que definirá chave estrangeira.
- **On Field** Nome do campo (na tabela-filha) ==> chave estrangeira).
- **FK Tables** Selecione o nome da tabela-mãe.
- **FK Field** Selecione o campo que é chave primária na tabela-mãe.

Siga os procedimentos conforme indicam as figuras 5.25a, 5.25b e 5.25c para criar a primeira chave estrangeira: FK_GenFilme, e depois as outras duas

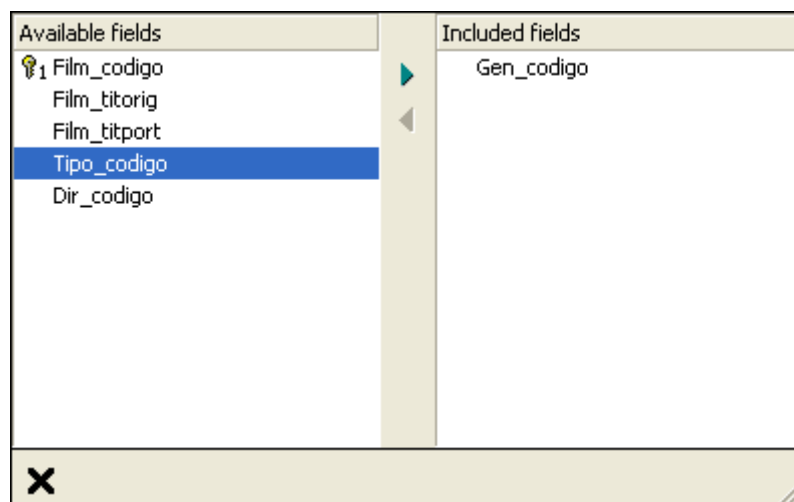


Figura 5.25a - Selecionando o campo da chave estrangeira

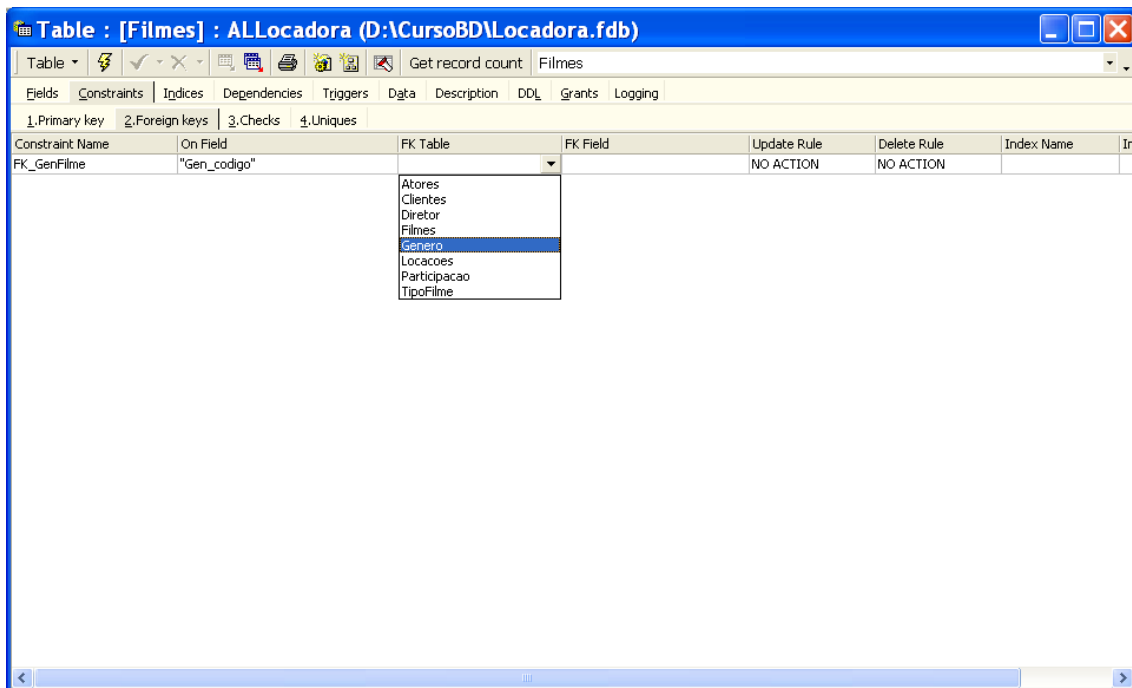


Figura 5.25b - Selecionando a tabela-mãe

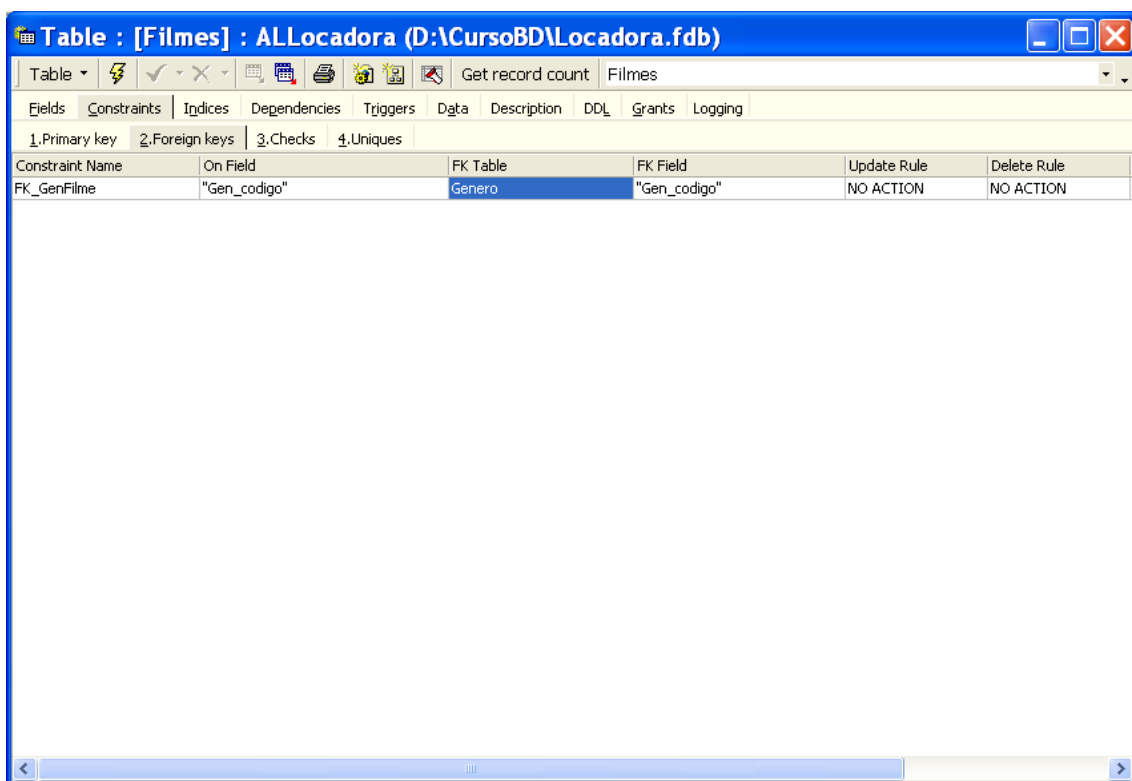



Figura 5.25c - A chave estrangeira "FK_GenFilme" pronta para ser criada

5º Passo - Após definir a chave estrangeira, acione o menu **Table/Compile** ou simplesmente dê um clique no ícone  (ou pressione CTRL+F9) para compilar. O resultado é a janela mostrada na **Figura 5.26** mostrando a instrução sql gerada no processo de criação da chave estrangeira "FK_GenFilme".

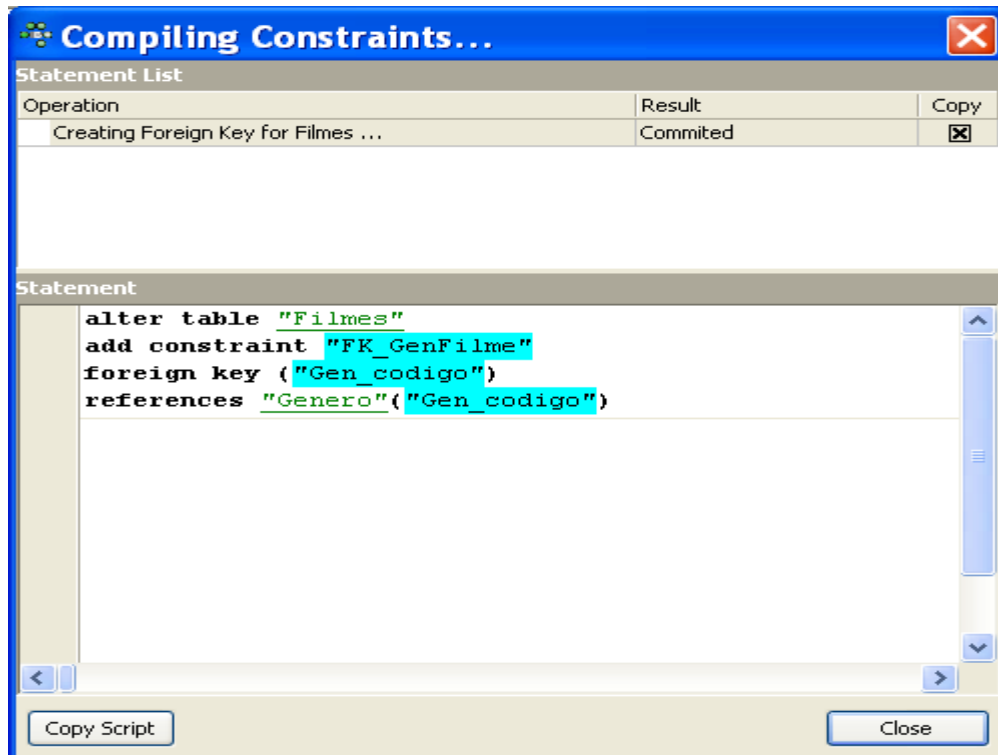


Figura 5.26 - Criando chave estrangeira na tabela "Filmes": 5º Passo

6º Passo - Clique no botão rotulado como **[Close]** da janela da **Figura 5.26** para encerrar o processo de criação da primeira chave estrangeira. Após isto o sistema emitirá um pequeno alerta sonoro indicando que o processo está concluído. Depois aparecerá uma janela com um botão **[Commit]**; clique nele para gravar a chave estrangeira no banco de dados. A **Figura 5.27** mostra a tabela com a FK criada.

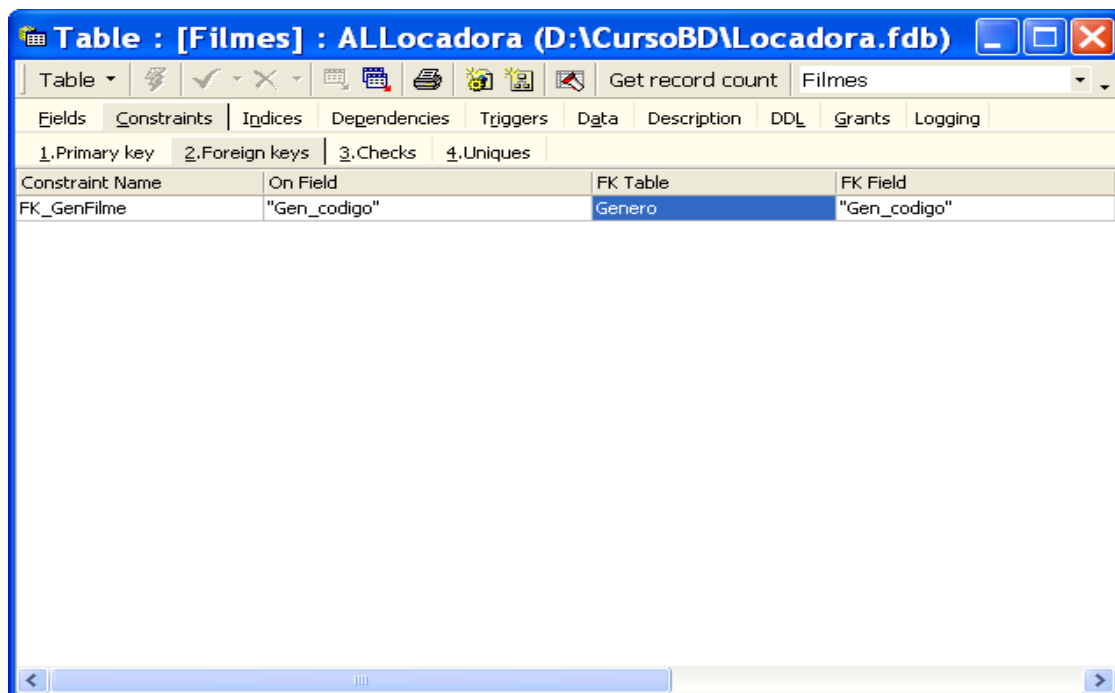

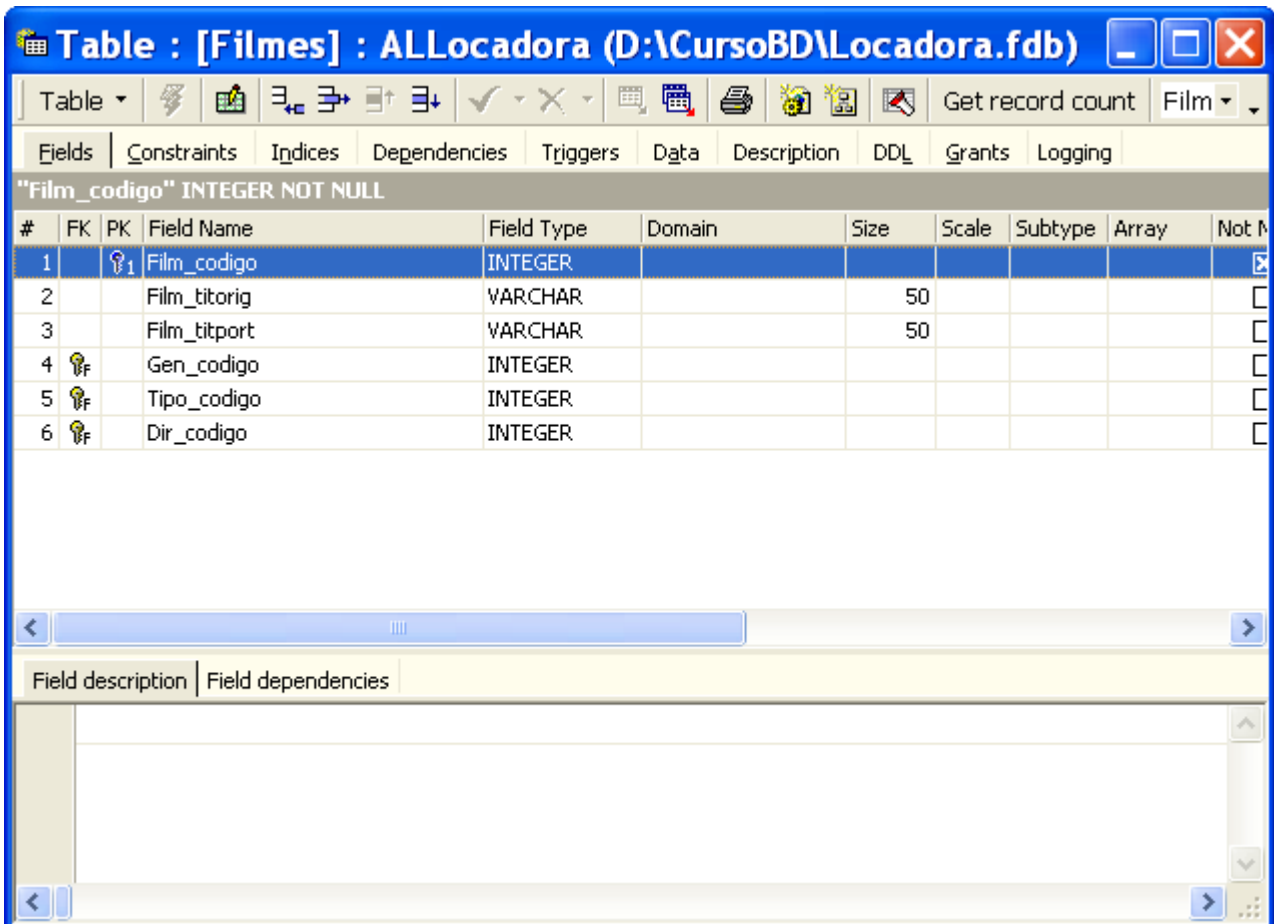


Figura 5.27 - A tabela "Filmes" exibindo uma de suas chaves estrangeiras

Assim, para criar as outras duas chaves estrangeiras basta repetir os passos 3, 4 e 5, conforme foi feito para a chave “FK_GenFilme”. Com um duplo clique sobre o nome da tabela “Filmes” na janela do “Database Explorer”, é possível visualizar a estrutura da tabela, agora exibindo as duas chaves estrangeiras (indicadas pelo ícone  ao lado dos três campos “Gen_codigo”, “Tipo_codigo” e “Dir_codigo”; confira na **Figura 5.28**. O mesmo corre em relação às outras tabelas que possuam alguma chave estrangeira. Assim, finalmente, na janela do “Database Explorer” (**Figura 5.29**) são exibidas todas as tabelas do banco e as chaves (índices).







#	FK	PK	Field Name	Field Type	Domain	Size	Scale	Subtype	Array	Not Null
1			Film_codigo	INTEGER						<input checked="" type="checkbox"/>
2			Film_titorig	VARCHAR		50				<input type="checkbox"/>
3			Film_titport	VARCHAR		50				<input type="checkbox"/>
4			Gen_codigo	INTEGER						<input type="checkbox"/>
5			Tipo_codigo	INTEGER						<input type="checkbox"/>
6			Dir_codigo	INTEGER						<input type="checkbox"/>

Figura 5.28 - A tabela “Filmes” exibindo uma as suas três chaves estrangeiras

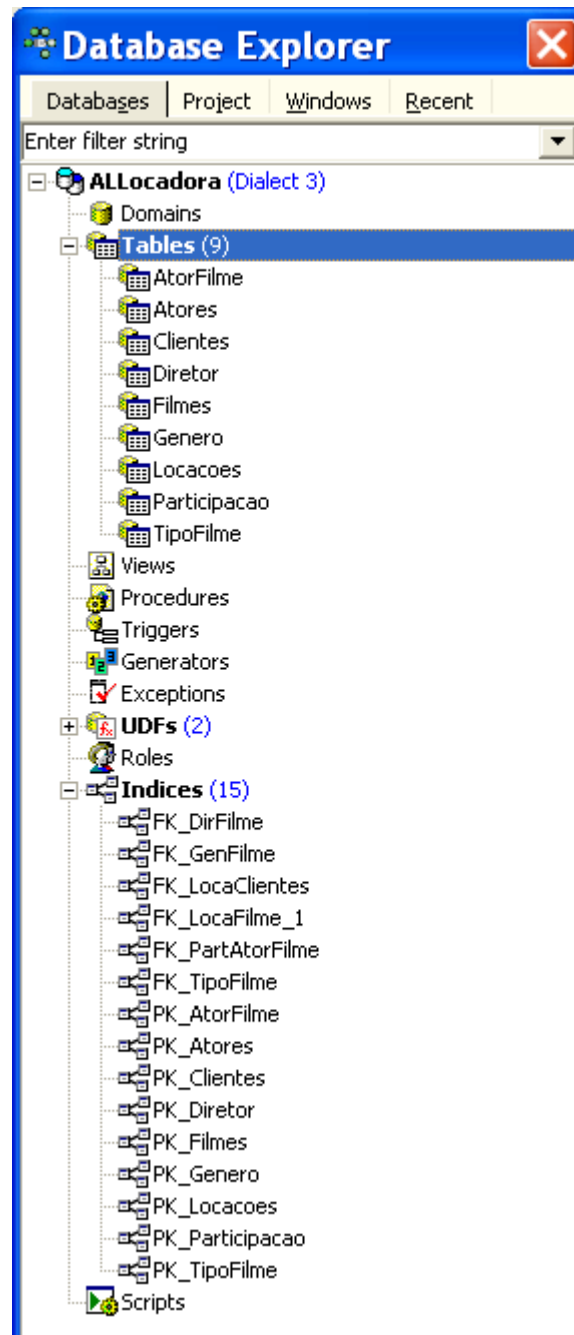


Figura 5.29 - O banco “Locadora” exibindo os índices de suas tabelas

5.6 - Criação de chaves secundárias com o IBExpert

Conforme já explicado em capítulos anteriores, as chaves secundárias (índices secundários) são muito importantes num banco de dados, pois permitem reorganizar os registros das tabelas de acordo com um campo pré-determinado de maneira muito rápida. A criação desses índices com o IBExpert é muito fácil, pois é feito interativamente, sem a necessidade de digitar explicitamente instruções sql.

Para ilustrar isto, vamos criar o índice secundário denominado “Ind_CliNome” que indexará o campo “Cli_nome” da tabela “Clientes”. O processo é muito parecido com a criação de chave estrangeira; a diferença é que não existem as figuras da tabela-mãe e tabela-filha, pois a criação de índices secundários não depende de nenhum outro tipo de chave, além de ser permitido criar várias chaves secundárias em vez de apenas uma.

1º Passo - Dê um duplo clique sobre “Clientes” na janela do “DB Explorer” e em seguida clique na guia “Index” (Índices em Português). O resultado é a janela da **Figura 5.30**, mostrando todos os índices da tabela até o momento; no caso apenas o índice primário “PK_Clientes”

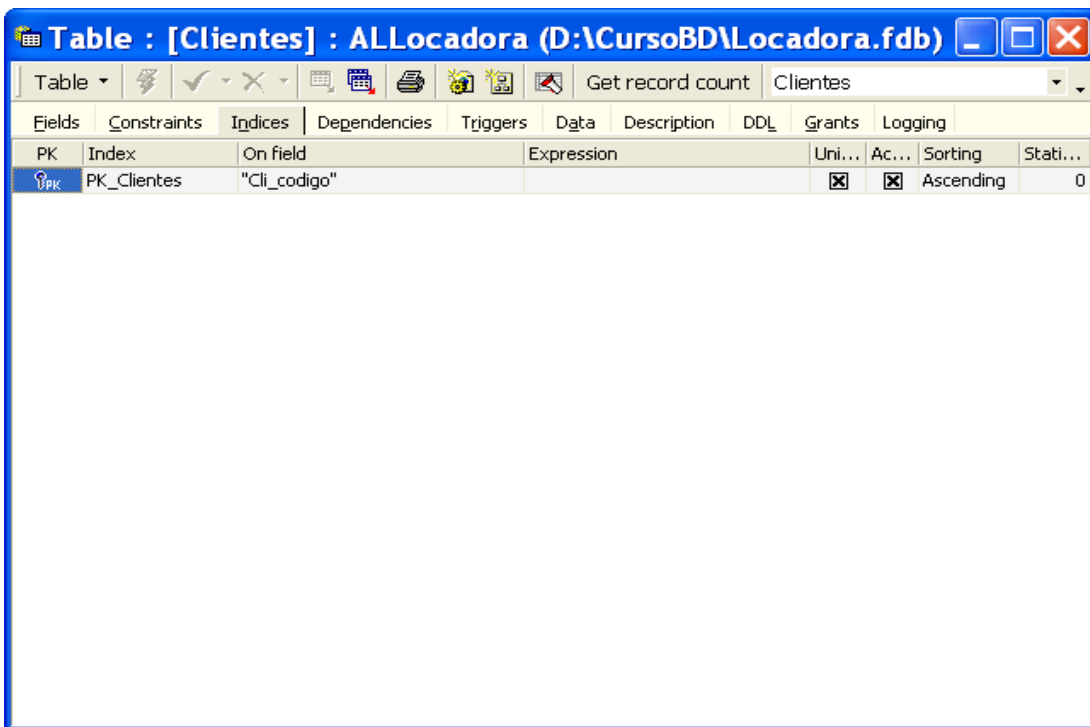


Figura 5.30 - Exibindo os índices atuais da tabela “Clientes”

2º Passo - Clique com o botão direito do *mouse* sobre a parte branca da janela e escolha a opção “New Index..” no *menu pop-up* que se apresenta; imediatamente será mostrada uma janela com um novo registro de índice (com o nome padrão sugerido de “Clientes_IDX1”); renomeie para **Ind_CliNome** e na coluna “On field” selecione o campo “Cli_nome” e mova-o para o lado direito da janela (Included fields); Vide **Figura 5.31a**.

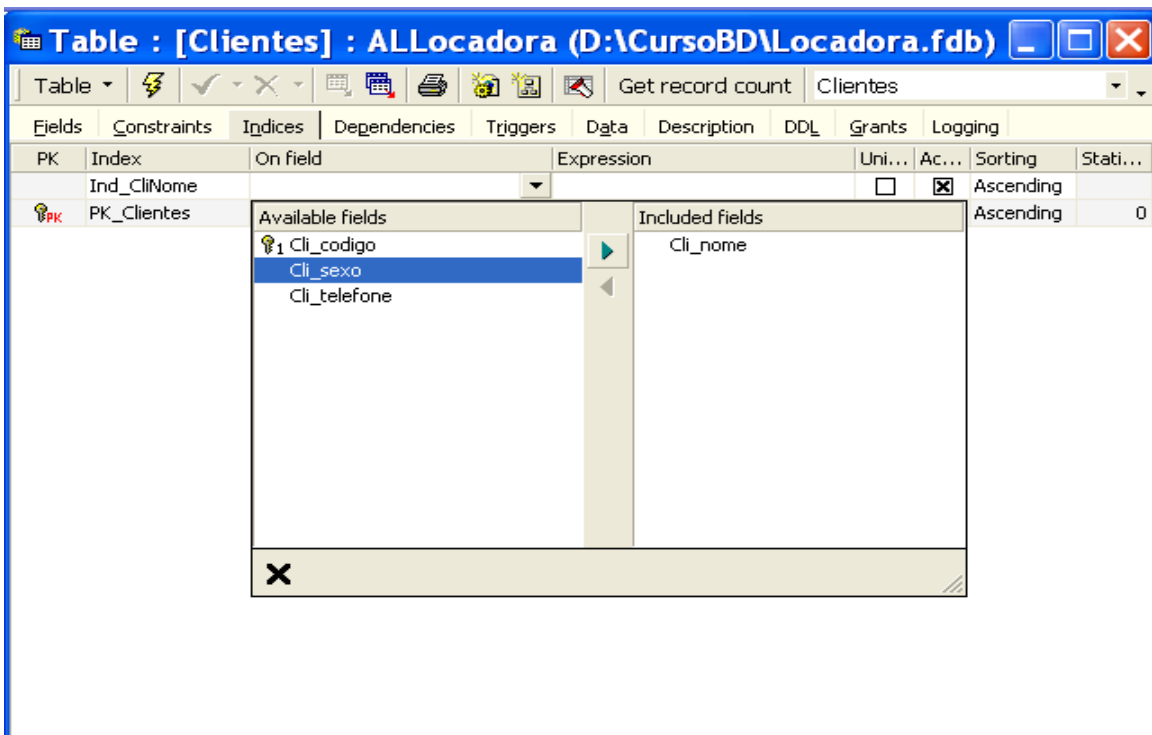


Figura 5.31a - Selecionando o campo que será indexado

3º Passo - Preencha os campos adequadamente:

- *Unique* (desmarcado)
- *Active* (marcado)
- *Sorting* **Ascending**

O resultado é mostrado na **Figura 5.31b**.

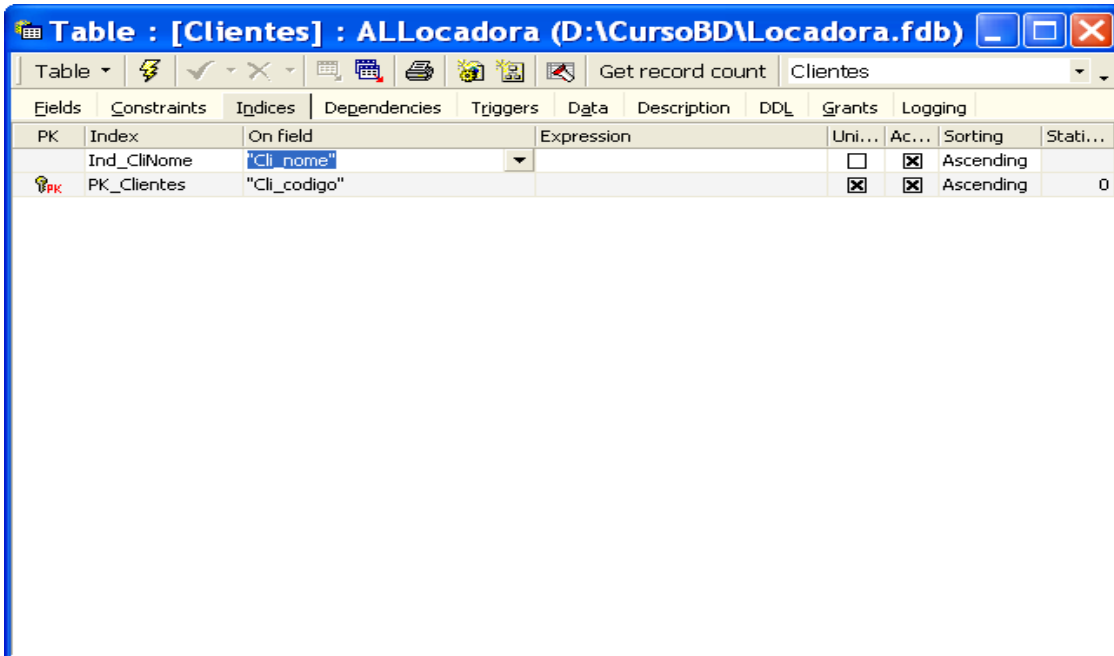


Figura 5.31b - Selecionando o campo que será indexado

4º Passo - Compile, acionando o *menu Tabela/Compile* ou simplesmente dê um clique no ícone do raio . O resultado é a janela da **Figura 5.32** mostrando a criação do índice "Ind_CliNome". Em seguida clique no botão **[Commit]** para efetivar a compilação.

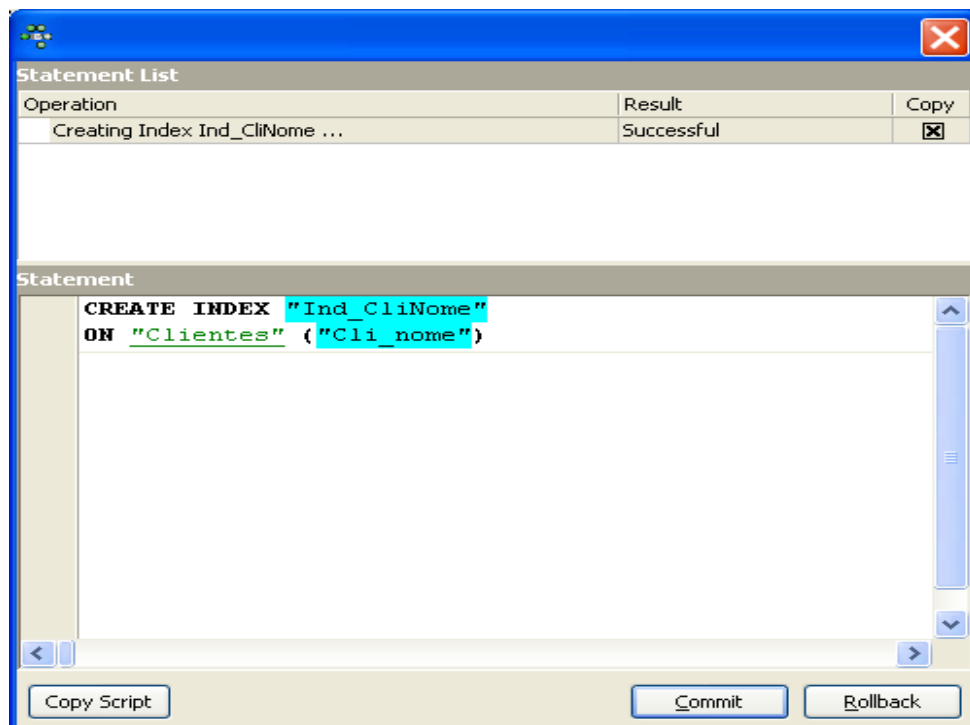


Figura 5.32 - Compilando para criar o índice "Ind_CliNome"

Observe (**Figura 5.33**) o banco “Locadora”: agora existe um índice secundário para a tabela “Clientes”: **Ind_CliNome** em destaque..

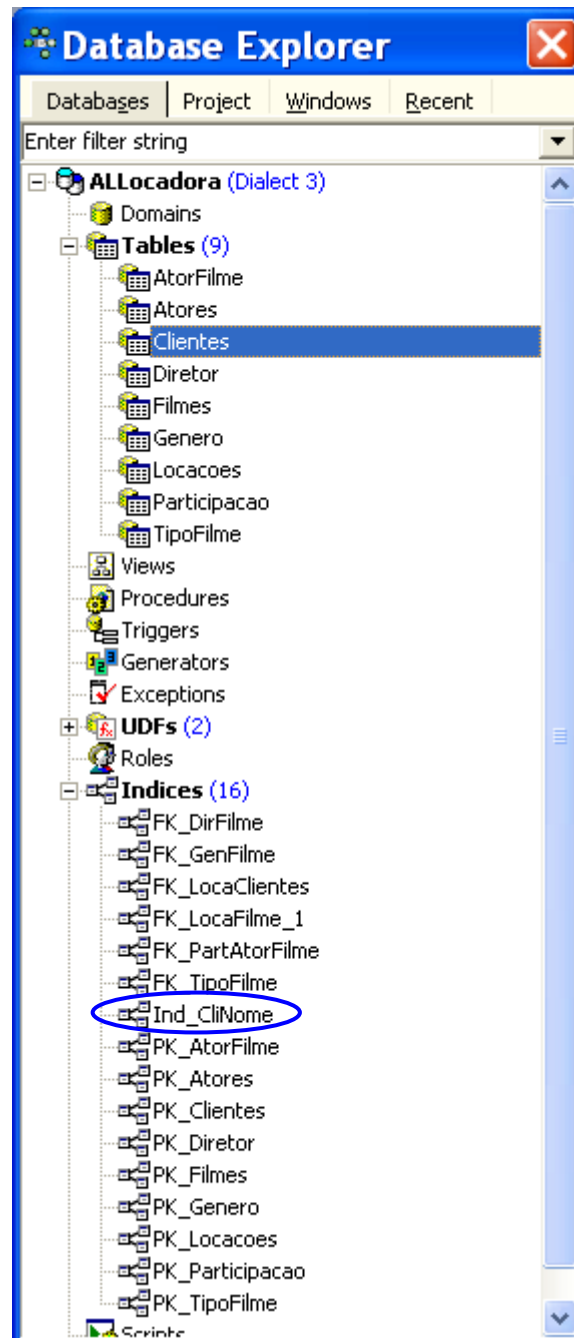


Figura 5.33 - Banco “Locadora” exibindo os índices de suas tabelas

5.7 - Trabalhando com Domínios

5.7.1 - Criação de Domínios manualmente

Quando criamos um novo campo de uma tabela, precisamos definir seu domínio (faixa de valores, além das operações permitidas). Também pode ser criado um novo domínio (*domain*) como um “modelo” de tipo de dado para validar entradas nos campos das tabelas. Uma vez criado, o domínio pode ser empregado para validar dados em várias tabelas de um mesmo banco. Por

exemplo, tabelas como “Clientes” e “Fornecedores” podem ter os campos “Nome”, “Endereco”, “Cidade”, “CEP”, “Estado”, “Fatura”, etc, validados com os mesmos domínios.

Para exemplificar, vamos criar um domínio que valide o sexo de um ator na tabela “Atores”. Para criar um domínio basta executar uma instrução com a seguinte sintaxe:

```
CREATE DOMAIN “Nome_do_domínio” AS Tipo_de_dado [CHECK(valor)]
```

A cláusula **CHECK** é usada para verificar se o valor entrado é compatível com o padrão estabelecido pelo usuário; por exemplo, verificar valores mínimos e máximos de tipos numéricos.

Mas, onde deve ser digitada essa instrução??!!

Conforme foi dito no início deste capítulo, a ferramenta IBExpert deve ser utilizada justamente para evitar que o usuário digite instruções sql. Mas, apenas como curiosidade, vamos digitar essa instrução e executá-la, mesmo com o IBExpert.

Para exemplificar o uso de domínios vamos criar na tabela “Atores” um domínio para validar o sexo do ator, denominado “DomSexo” quer vai validar a entrada do sexo como “F” ou “M”, apenas essas duas letras (maiúsculas). A instrução para isto é a seguinte:

```
CREATE DOMAIN “DomSexo” CHAR(1) CHECK (Value IN ('F', 'M'));
```

- 1) Conecte o banco de dados, clique na opção **Tools** na barra do *menu* principal do IBExpert e selecione a opção “SQL Editor”, conforme está mostrando a **Figura 5.34a**.
- 2)

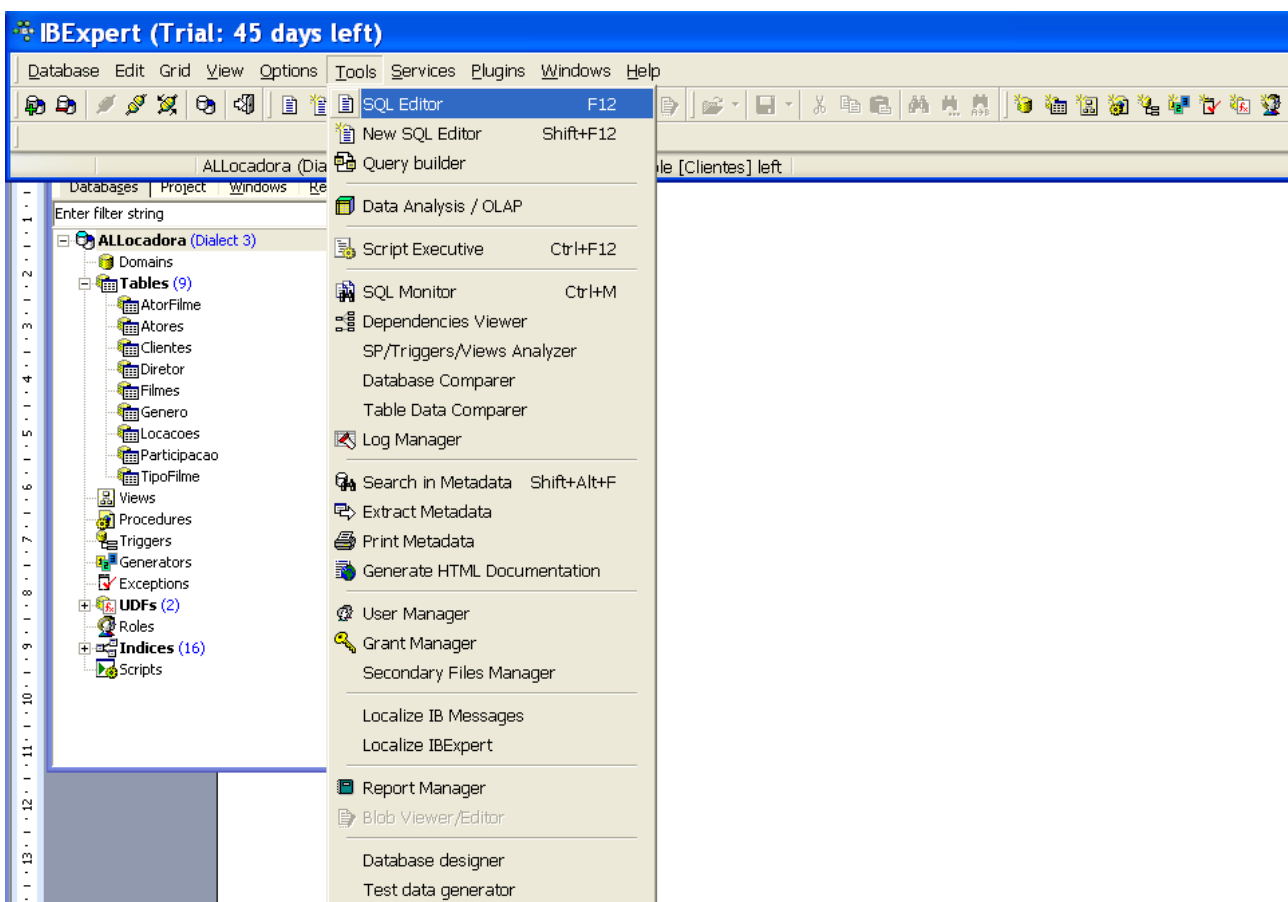


Figura 5.34a - Acionando o editor de instruções sql do IBExpert

2) Na janela da **Figura 5.34a** confirme a opção “SQL Editor” para entrar no editor de instruções sql do IBEExpert. O resultado é a janela da **Figura 5.34b** (em branco), esperando que seja digitada uma instrução sql.

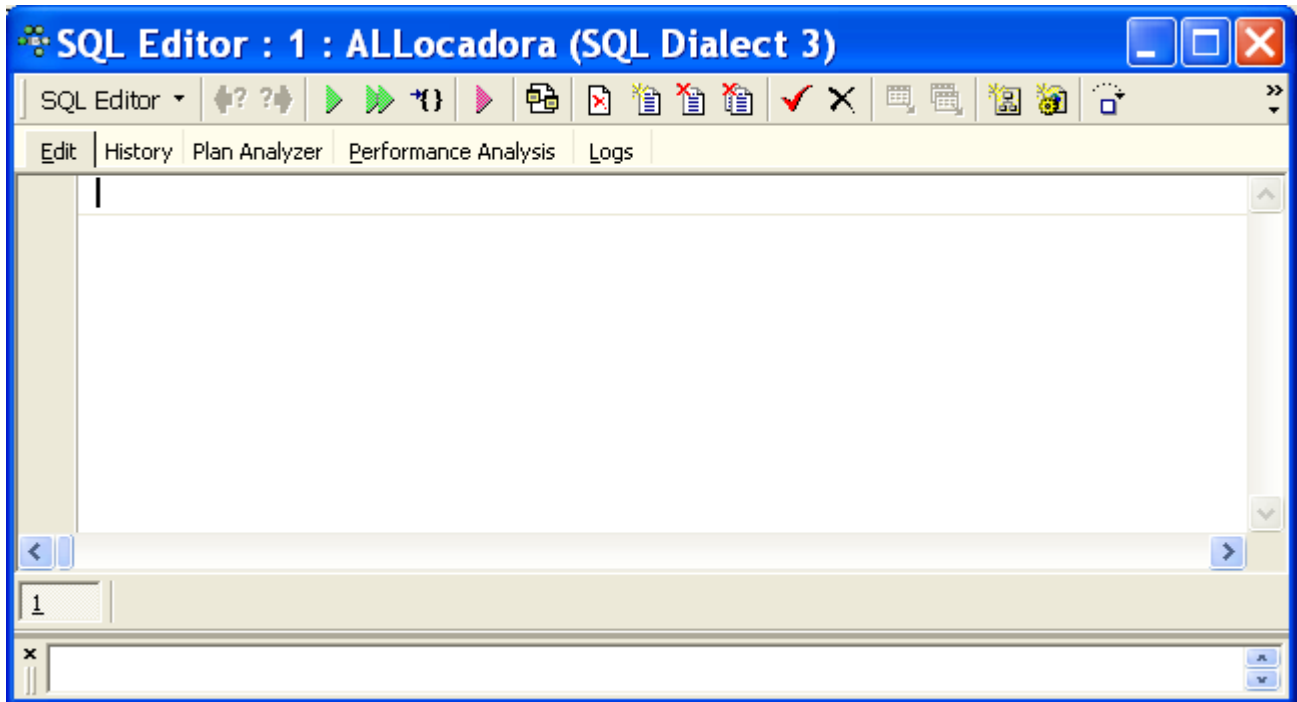


Figura 5.34b - O “SQL Editor” pronto para receber uma nova instrução

3) Digite a instrução a partir do ponto em que se encontra o cursor *piscante*. Veja como fica na **Figura 5.34c**.

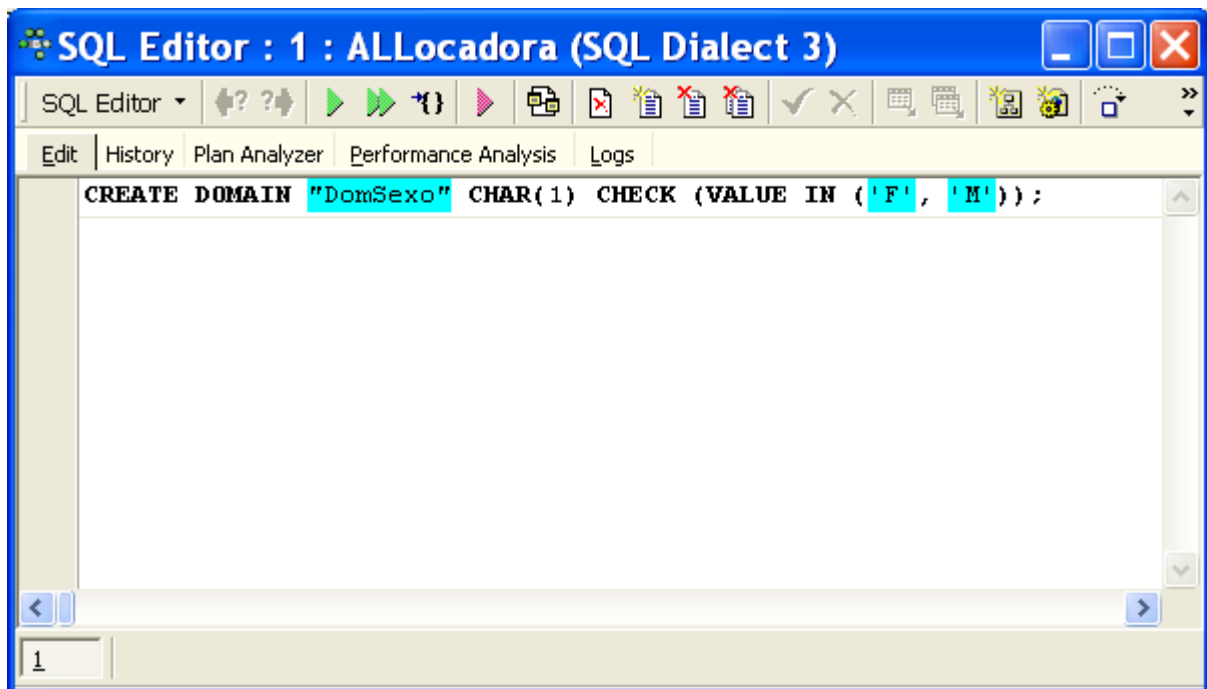



Figura 5.34c - A instrução sql digitada no “SQL Editor”

4) Para criar, efetivamente, o domínio através da instrução sql digitada no Editor, deve-se executar a instrução, acessando **SQL Editor/Executar** ou clicando diretamente no ícone de execução . O resultado é mostrado na **Figura 5.34d**.

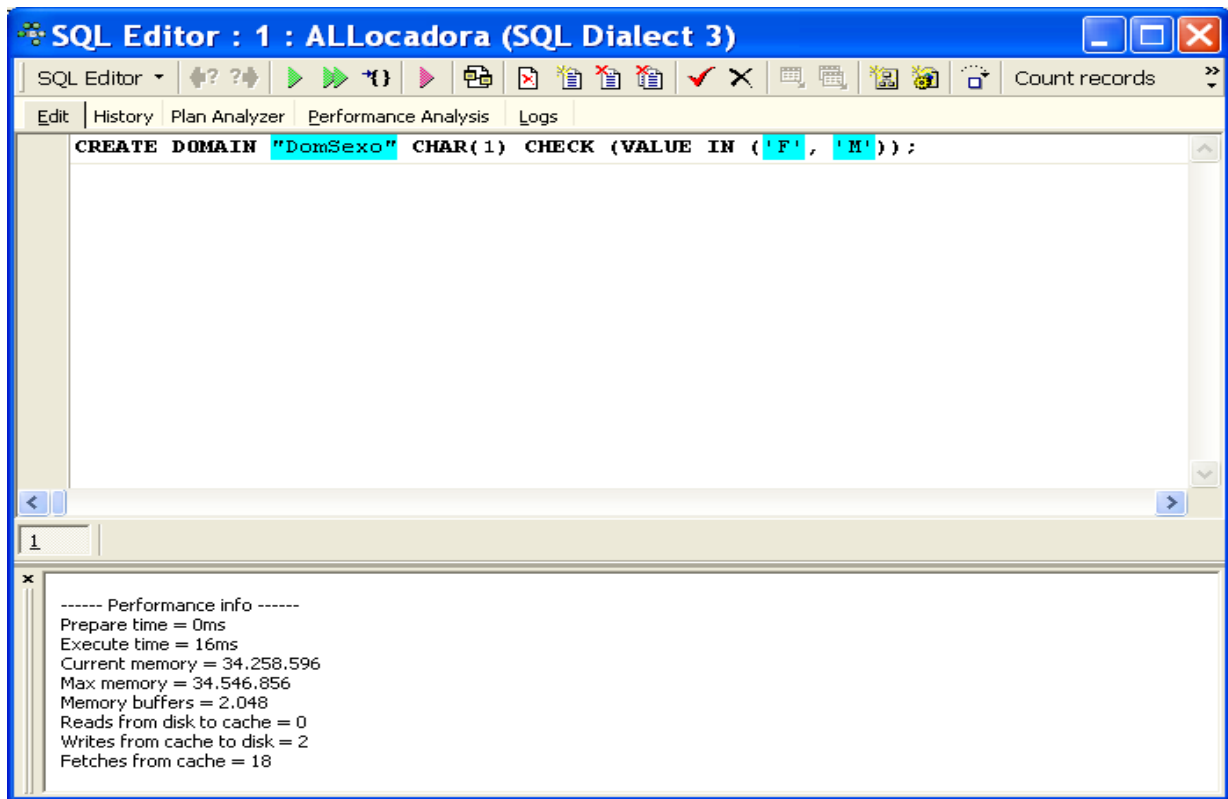



Figura 5.34d - A instrução sql sendo executada

5) Finalmente a operação deve ser *commitada* clicando no ícone . Após a emissão de um alerta sonoro, indicando que o processo terminou, aparece a **Figura 5.34e**, mostrando que a transação foi *commitada* corretamente. Em seguida é só fechar a janela...

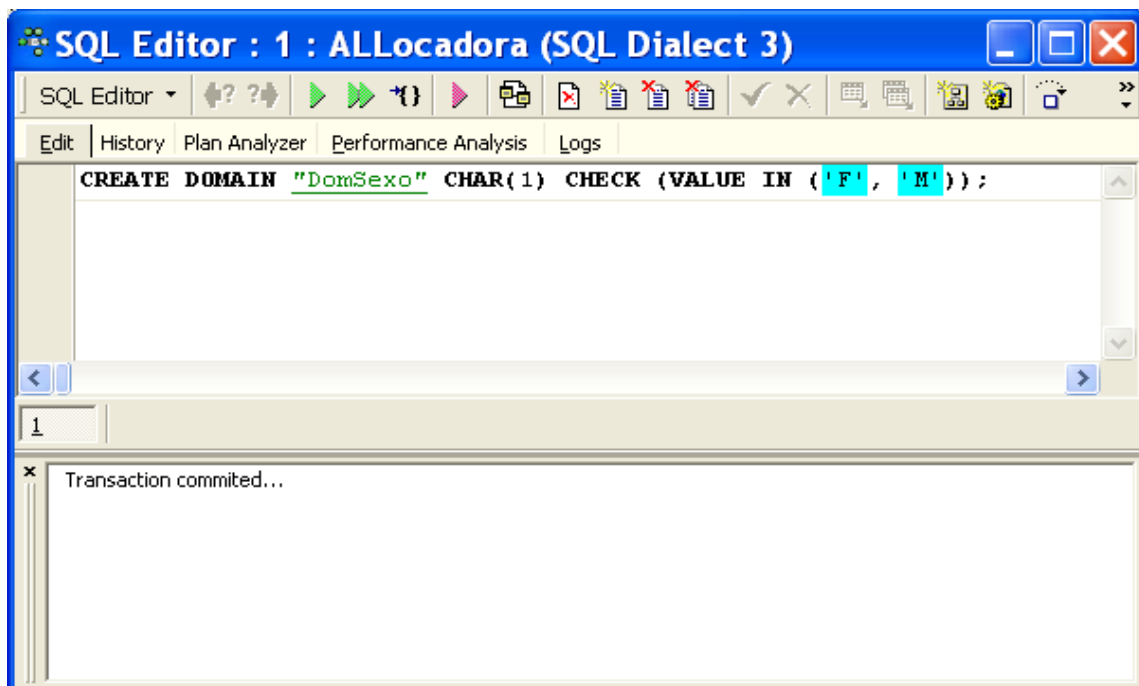


Figura 5.34e - A instrução sql foi *commitada* ...

A **Figura 5.35** mostra o nosso banco de dados, agora exibindo o domínio criado. Observe que esse domínio está no mesmo nível que as tabelas; portanto, pode ser utilizado por qualquer uma delas, pois não é exclusivo de nenhuma tabela.

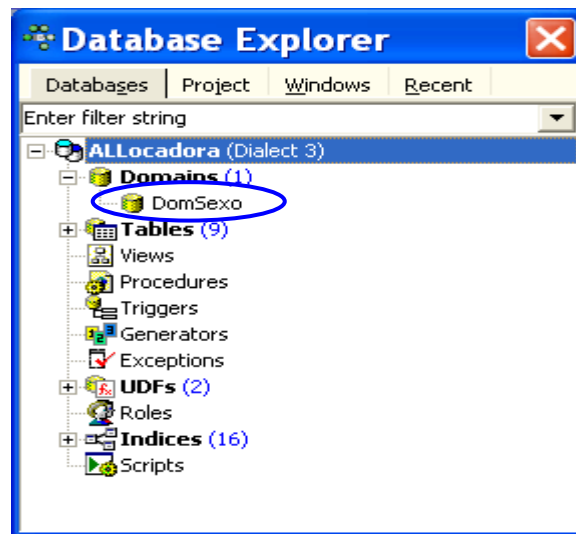


Figura 5.35 - Exibindo o novo domínio criado

5.7.2 - Criação de Domínios interativamente

No item anterior criamos o domínio “DomSexo” de maneira manual, através da execução de uma instrução sql digitada no “SQL Editor” do IBExpert. Neste item vamos criar o mesmo domínio, agora de maneira interativa.

1º) Na janela do “DB Explorer” clique com botão direito do *mouse* sobre “Domains”, e no *menu pop up* que aparece selecione a opção “New Domain”, como mostra a **Figura 5.36a**.

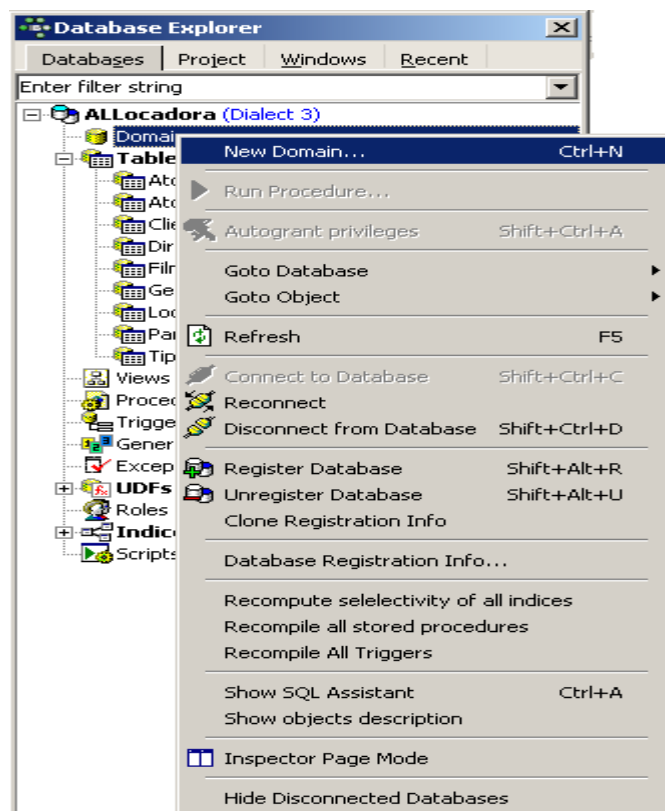


Figura 5.36a - Criando um domínio interativamente: Passo 1

2º) Ao efetivar a seleção de criação de novo domínio no passo anterior, aparecerá a janela da **Figura 5.36b**, pronta para a definição desse novo domínio.

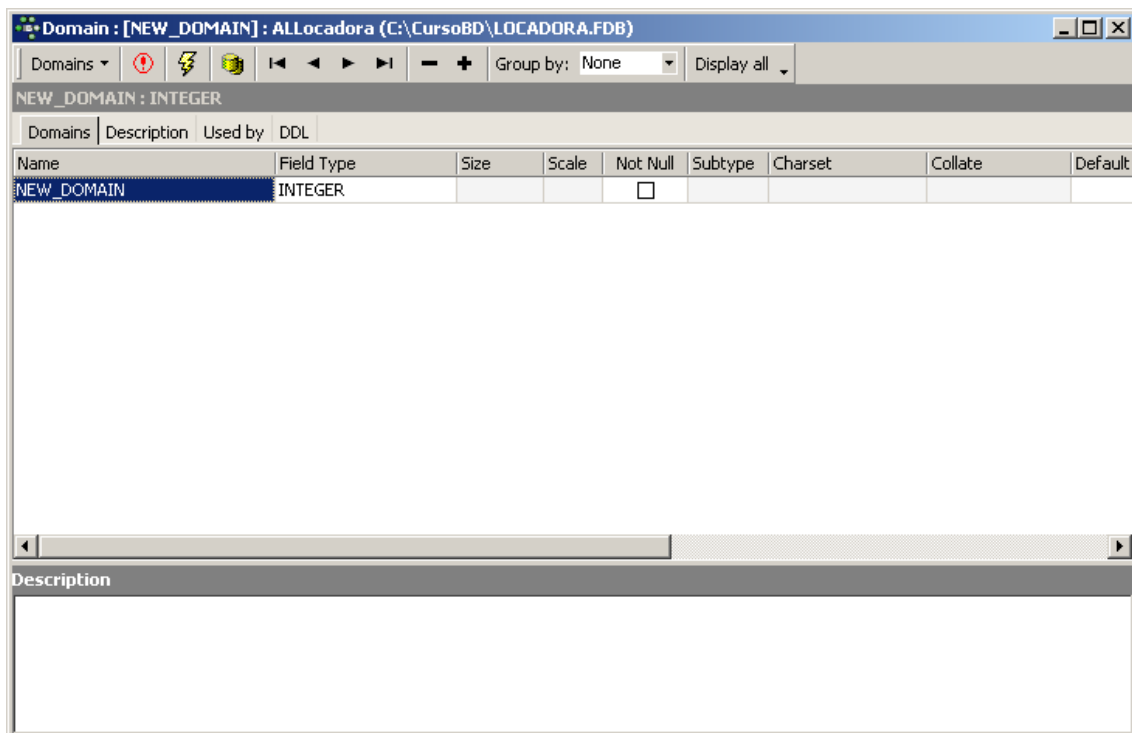


Figura 5.36b - Criando um domínio interativamente: Passo 2

3º) Preencha os dados para a criação do domínio, como mostrado na **Figura 5.36c**. Observe a informação que deve ser colocada na coluna "CHECK": **(VALUE IN('F', 'f', 'M', 'm'))**. Essa informação alerta ao sistema que só serão aceitas as letras "efe" e "eme" (minúsculas ou maiúsculas), agora permitindo, também, letras minúsculas para o sexo. Se for tentado entrar com qualquer outro caracter que não um desses quatro acima descritos, ocorrerá um erro e a entrada não será aceita.

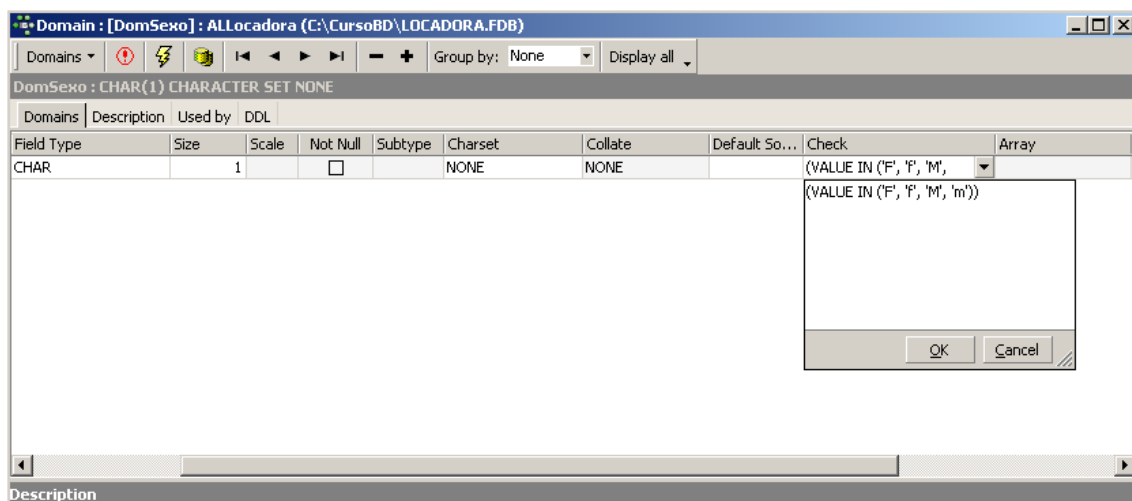


Figura 5.36c - Criando um domínio interativamente: Passo 3

4º) Confirme os dados da janela da **Figura 5.36c** para o novo domínio. Após esta confirmação, a janela da **Figura 5.36d** aparecerá.

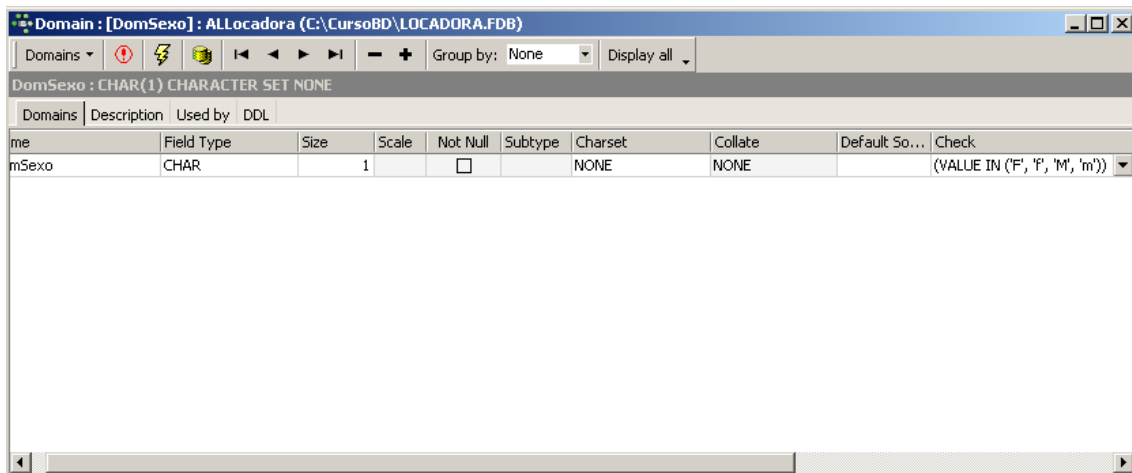


Figura 5.36d - Criando um domínio interativamente: Passo 4

5º) Depois de confirmar os dados do novo domínio, compile-o, clicando no ícone

O resultado é a janela da **Figura 5.36e**, mostrando o resultado da compilação com sucesso). Caso houvesse algum erro na definição do domínio, uma mensagem de erro (em vermelho) apareceria nessa janela.

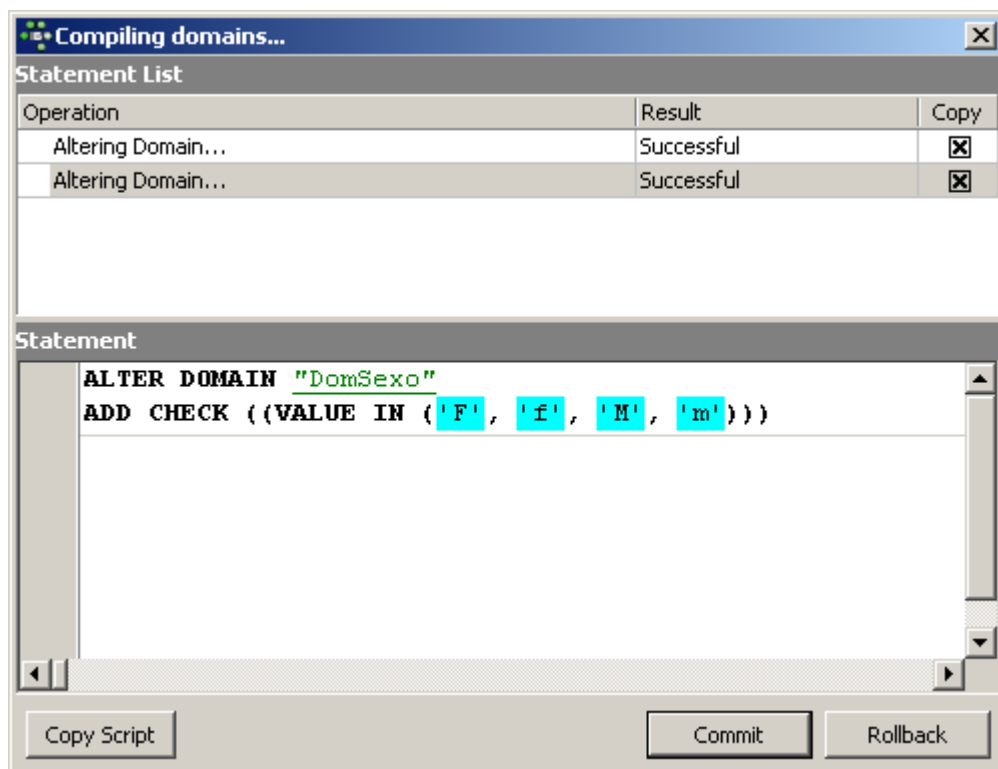


Figura 5.36e - Criando um domínio interativamente: Passo 5

6º) E para finalizar, basta clicar no botão **[Commit]** da janela da **Figura 5.36e**. Essa ação grava, efetivamente, o novo domínio banco de dados. O resultado pode ser visto na **Figura 5.37**, onde o “DB Explorer” exibe o novo domínio incorporado ao banco de dados, tal como já havia sido feito manualmente no item anterior.

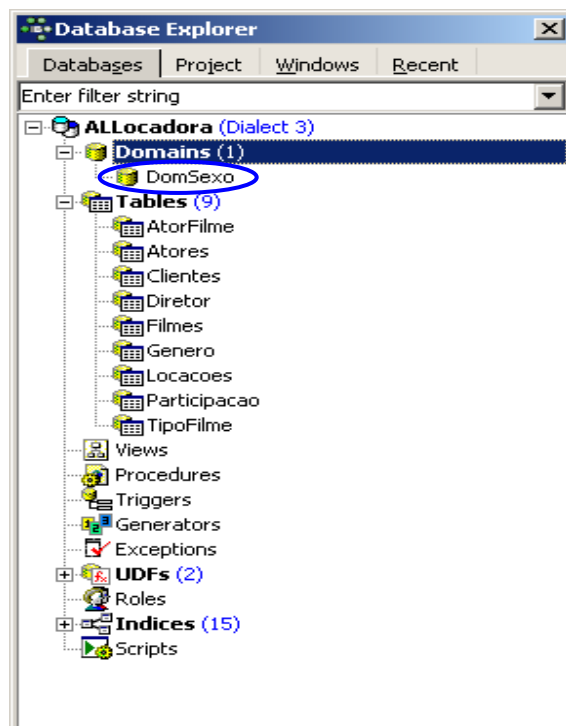


Figura 5.37 - O novo domínio agregado ao banco de dados

Observe nesses seis passos executados, que o segredo para criar um domínio está no controle e definição da cláusula “CHECK”; é ela que comanda o processo de validação do dado no domínio.

5.7.3 - Testando o domínio criado

Considerando o domínio criado anteriormente (DomSexo), vamos testar sua eficiência editando o campo “Ator_sexo”. Lembre-se de que quando a tabela “Atores” foi criada, o campo “Ator_sexo” tinha seu domínio definido como “Char(1)”; isto significa que QUALQUER caracter de tamanho 1 poderia ser usado para representar o sexo de um ator: X, x, Y, y, Z, H, 2, 4, 5, 9, 7, 3, etc). Agora, utilizando esse novo domínio, isto não mais poderá acontecer, caso o sexo não seja representado por “F”, “f”, “M”, ou “m”. Para testar isso, vamos tentar cadastrar o ator escocês “Sean Connery”.

1) Dê um duplo clique na tabela “Atores” mostrada no “DB Explorer”; o resultado é a exibição dessa tabela na sua forma estrutural, como mostra a **Figura 5.38**.

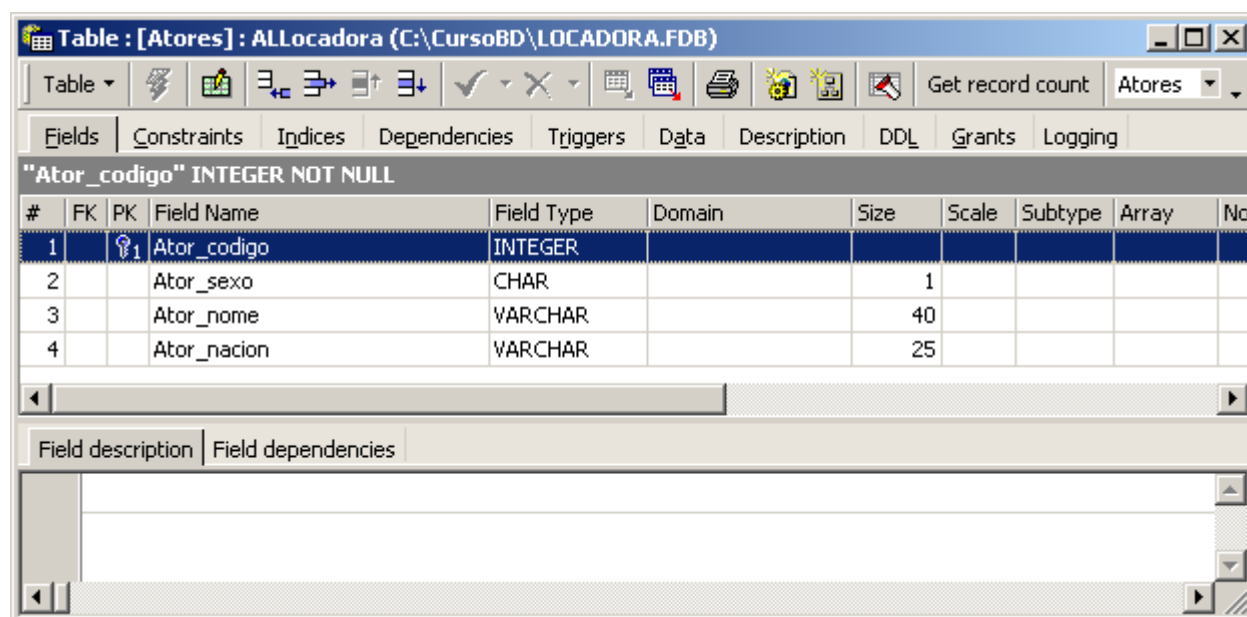


Figura 5.38 - A tabela “Atores” no modo estrutura pronta para edição dos campos

2) Clique com o botão direito o *mouse* sobre o campo “Ator_sexo” e selecione opção de editar campo “Edit Field Ator_sexo”, como mostra a **Figura 5.39a**. Observe nessa figura que a janela exibida também contém outras opções: de *dropar* (extinguir) campos, reordenar campos e copiar campos...

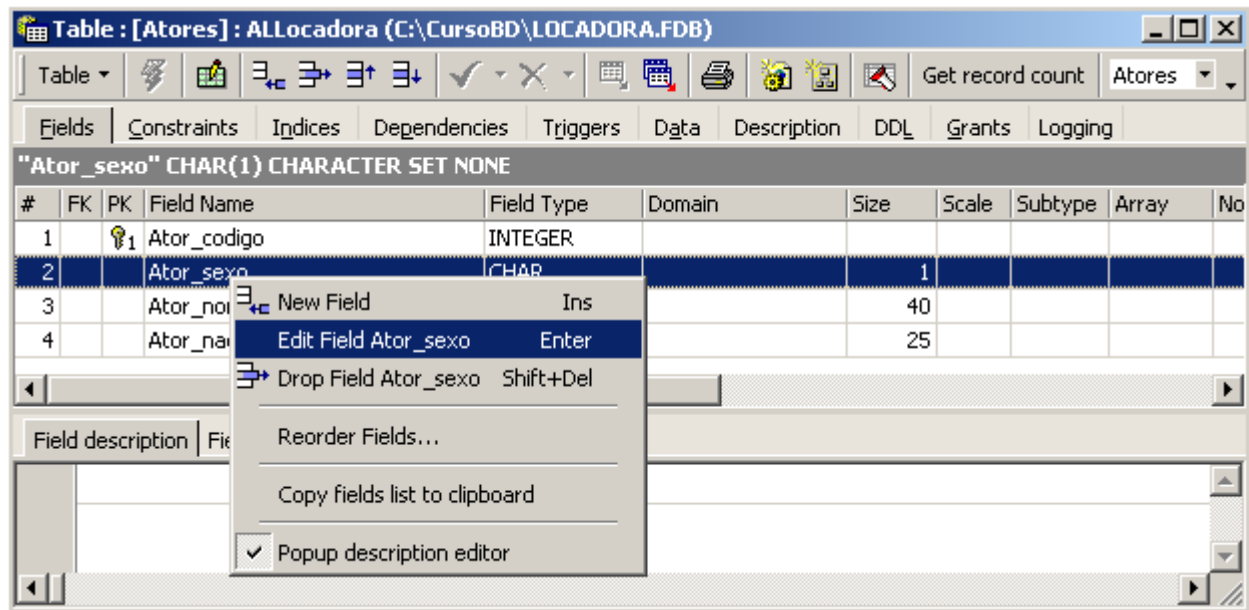


Figura 5.39a - Selecionando a opção para editar o campo “Ator_sexo”

3) Confirmando a seleção da opção “Edit Field Ator_sexo” na janela da **Figura 5.39a**, uma nova janela é exibida para que se faça efetivamente a alteração (edição) do campo marcado. Nessa nova janela abra a caixa *combo* “Domain” e lá estará o nosso domínio. Veja na **Figura 5.39b**.

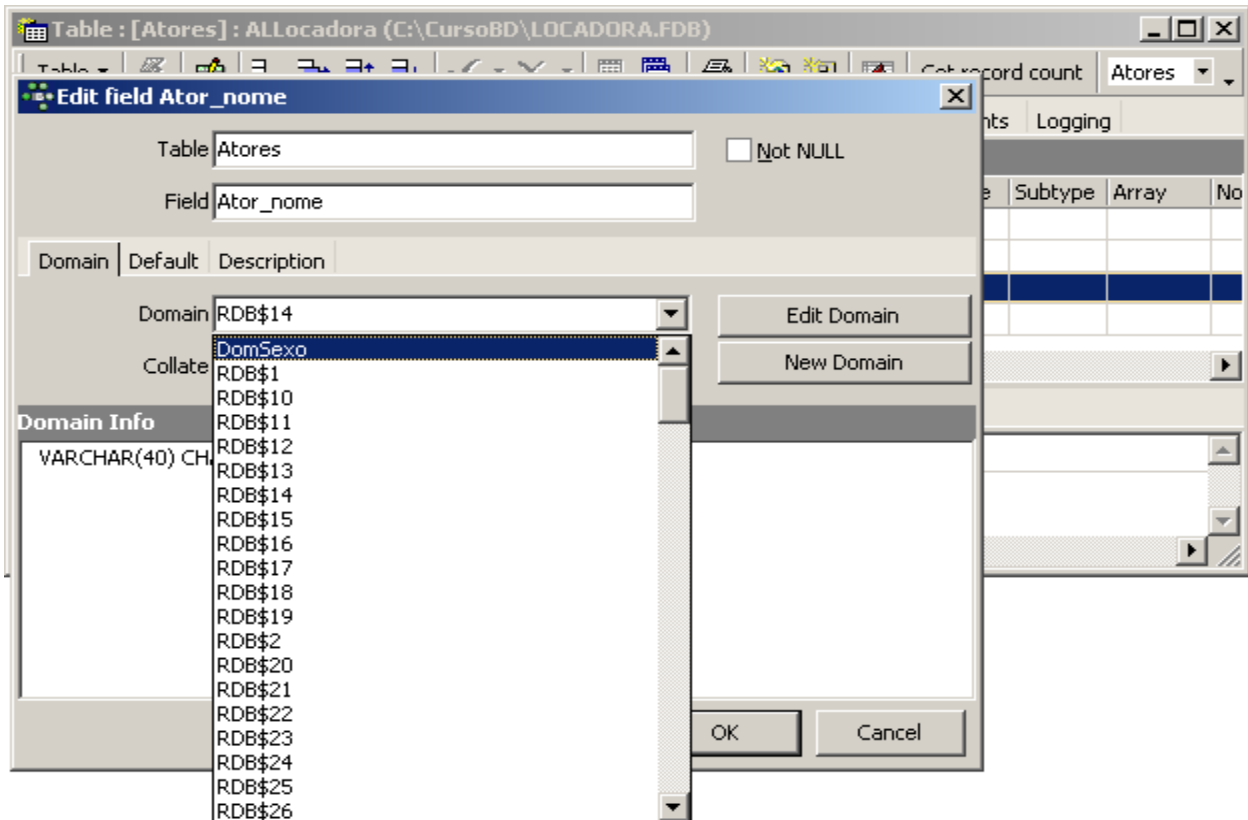


Figura 5.39b - Selecionando o novo domínio

4) Clique no botão **[OK]** da janela da **Figura 5.39b** para confirmar a escolha do novo domínio. O resultado é a exibição da janela com a alteração desejada para o **commit**. Vide **Figura 5.40**.

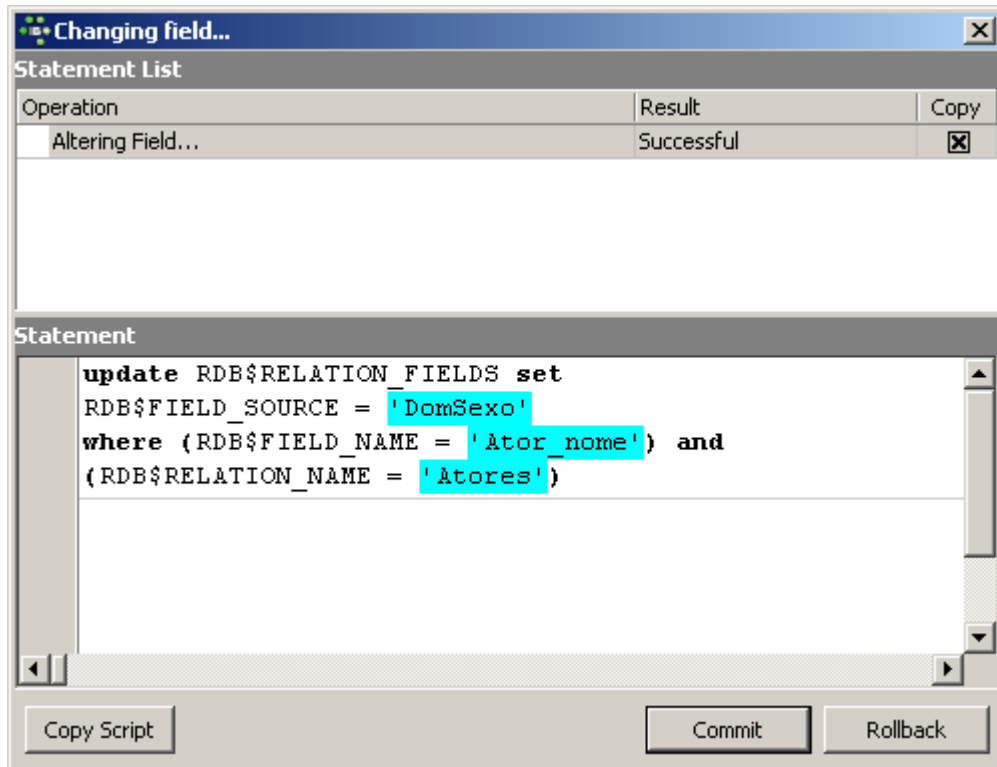


Figura 5.40 - Janela de alteração do domínio

5) Clique no botão **[Commit]** para gravar as alterações no banco de dados. O resultado é mostrado na **Figura 5.41**, que agora pode ser vista a nova estrutura da tabela “Atores”, onde o domínio do campo “Ator_sexo” passou a ser “DomSexo”; antes tinha tipo de dado, mas não um domínio definido.

#	FK	PK	Field Name	Field Type	Domain	Size	Scale	Subtype	Array	No
1		1	Ator_codigo	INTEGER						
2			Ator_sexo	VARCHAR	DomSexo	40				
3			Ator_nome	VARCHAR		40				
4			Ator_nacion	VARCHAR		25				

Figura 5.41 - A nova estrutura da tabela “Atores” com um novo domínio para o campo “Ator_sexo”

Agora, como teste, vamos cadastrar um ator na tabela “Atores”. Para isto, usaremos a mesma janela da **Figura 5.41**, clicando na guia “Data”. Isto faz com que a tabela se apresente no modo “folha de dados”, pronta para receber novos registros. Observe como fica na **Figura 5.42a**.

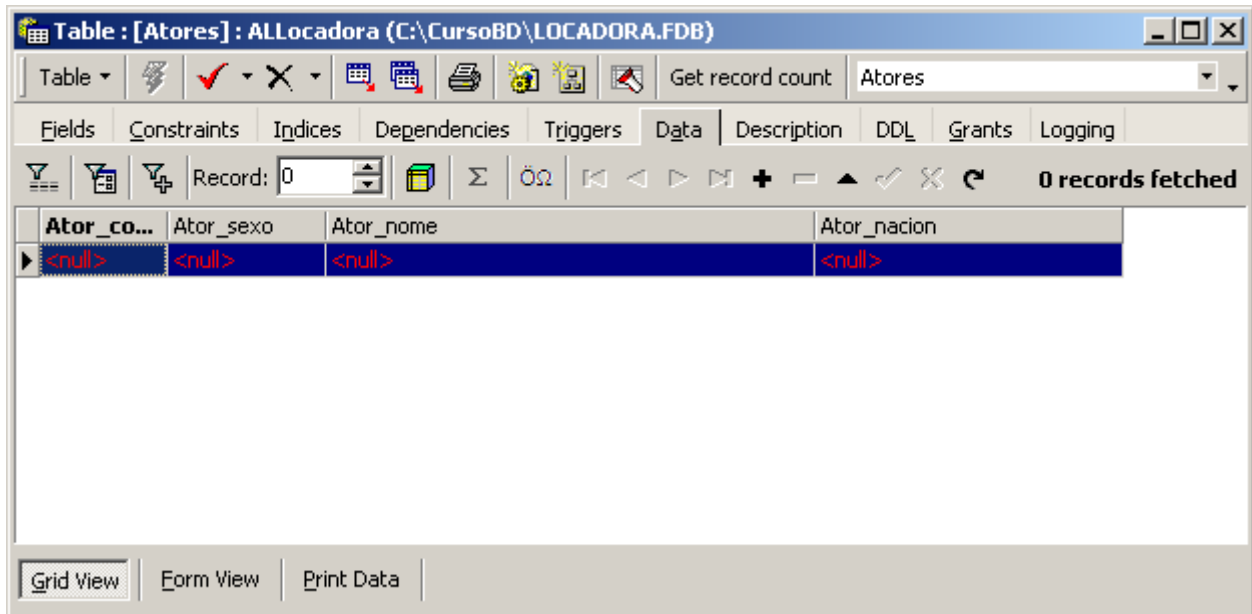


Figura 5.42a - A tabela “Atores” no modo *folha de dados* pronta para receber novos registros

Vamos, então, digitar os dados do ator “Sean Connery”, conforme foi dito no início deste item. Observe a entrada dos dados na **Figura 5.42b**.

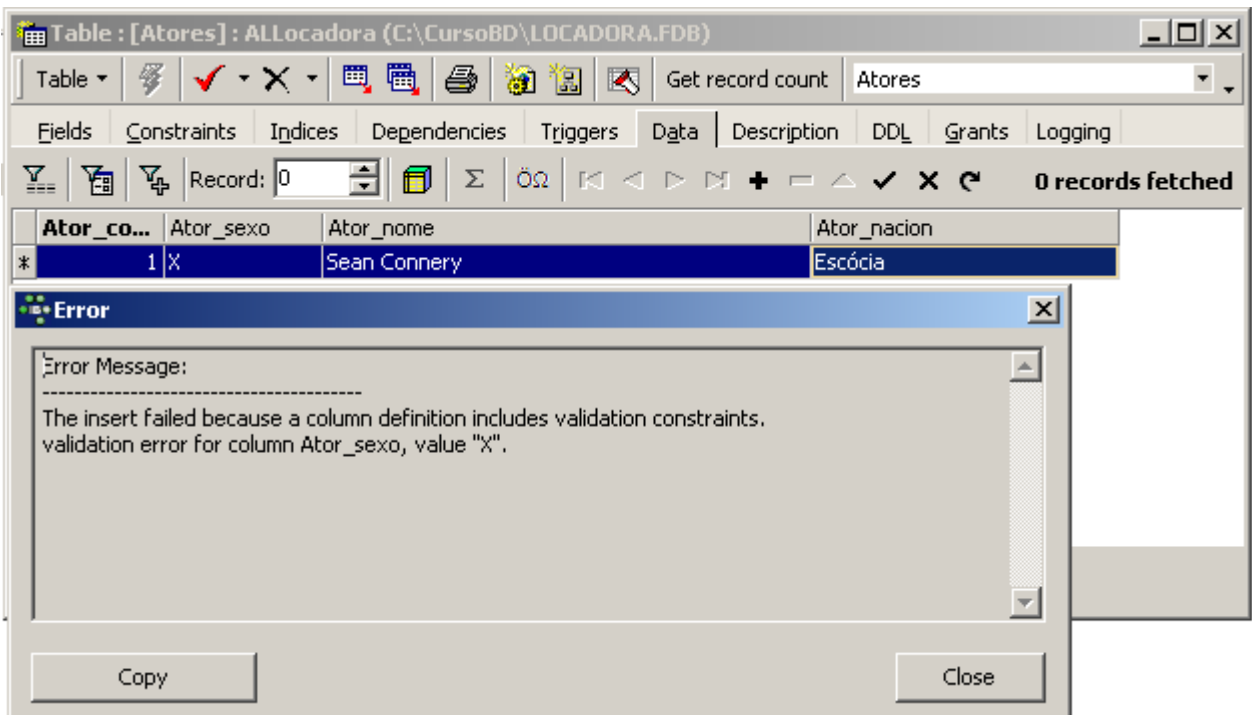



Figura 5.42b - Entrada de dados rejeitada: validada pelo domínio “DomSexo”

Note a **Figura 5.42b**; o sistema não aceitou a entrada de “X” para o sexo do ator e gerou uma mensagem de erro. Essa ação de validação foi feita pelo domínio “DomSexo”, que prevê apenas os caracteres “F”, “f”, “M” ou “m”. Agora, entrando com “M” (ou “m”) e confirmando no ícone de *commit*  o sistema aceitará, como demonstrado na **Figura 5.42c**.

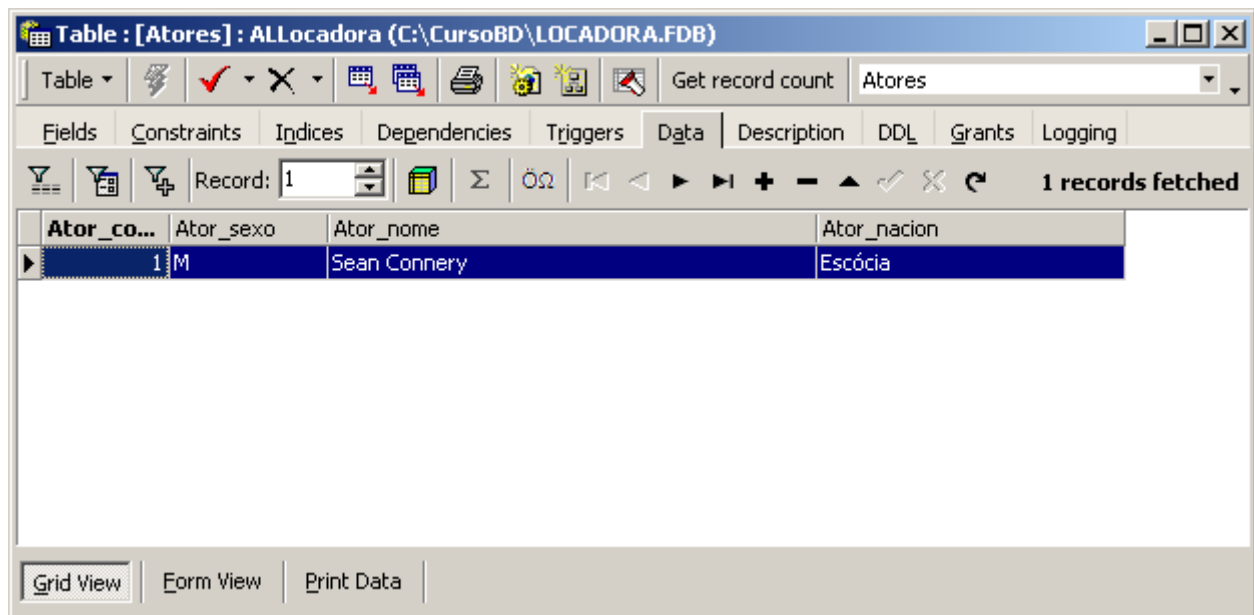


Figura 5.42c - Entrada de dados aceita para o campo “Ator_sexo”

5.8 - Trabalhando com Triggers

Triggers são “gatilhos” que podem ser disparados a partir da execução de uma instrução escrita numa tabela, para validar entradas de dados. Esses elementos são muito comuns em sistemas de informações em que algumas decisões precisam ser tomadas em função de algum evento ocorrido no bando de dados. Embora a *aplicação cliente* possa fazer o mesmo trabalho, o banco de dados fazendo isso pode diminuir o tráfego de informações na rede, além de não depender de instruções em linguagem de programação pura na aplicação. Por exemplo, a verificação do nível do estoque de alguma mercadoria; nesses casos o próprio banco pode avisar a necessidade de repor o estoque. Para os bancos de dados cliente-servidor essa tarefa é facilitada pelo emprego deste recurso.

Triggers são blocos de instruções do tipo “Transact-SQL”, automaticamente executados quando comandos **INSERT**, **DELETE** e **UPDATE** da linguagem SQL são executados, afetando os dados de uma tabela. Esses blocos de instruções são empregados para executar tarefas sob determinadas condições; eles são imediatamente solicitados quando algum evento ocorre sob condições pré-determinadas; por exemplo, restrições de acesso, consistências de dados, etc. Quando ocorre um evento desses tipos, a aplicação passa o controle para a *trigger* que dispara imediatamente seu bloco de instruções, tratando as exceções de maneira adequada. Uma *trigger* tem sintaxes diferentes para cada ação sobre ela; as duas mais básicas são de criação e de alteração.

1 - Criação de uma Trigger

A sintaxe geral de criação de uma *trigger* é a mostrada a seguir...

```
CREATE TRIGGER “Nome_da_trigger” FOR Tabela|View
ACTIVE BEFORE|AFTER INSERT POSITION 0
AS
begin
/* Bloco de instruções */
end
```

Entre **begin** e **end** é que são digitadas as instruções a serem disparadas (ativadas – antes ou depois) do evento “INSERT” ocorrer...

2 - Alteração de uma Trigger

```
ALTER TRIGGER "Nome_da_trigger" FOR Tabela|View
ACTIVE BEFORE|AFTER INSERT POSITION 0
AS
begin
/* Bloco de instruções */
End
```

3 - Exclusão de uma Trigger interativamente

```
DROP TRIGGER "Nome_da_trigger"
```

Os parâmetros envolvidos nas instruções de criação e alteração de uma *trigger* são os seguintes:

- Tabela | VIEW (obrigatório): Pode ser uma tabela ou uma *view*;
- ACTIVE | INACTIVE (opcional): Especifica a *trigger* ativa ou inativa;
- BEFORE | AFTER (obrigatório): Especifica se a *trigger* será disparada antes ou depois das operações INSERT, DELETE, UPDATE;
- DELETE | INSERT | UPDATE (obrigatório): Especifica qual a operação da tabela vai disparar a *trigger*.

Para trabalhar corretamente com *triggers* é necessário, antes, criar a exceção (**Exception**) que será gerada quando ocorrer algum erro de entrada de dados, emitindo uma mensagem adequada ao usuário.

4 - Criação de uma Exception interativamente

Para criar uma *exception* basta fazer o seguinte:

4.1) Dar *login* no banco;

4.2) Clicar com o botão direito do mouse sobre "Exceptions" no "DB Explorer"; a janela que se apresenta é a mostrada na **Figura 5.43a**.

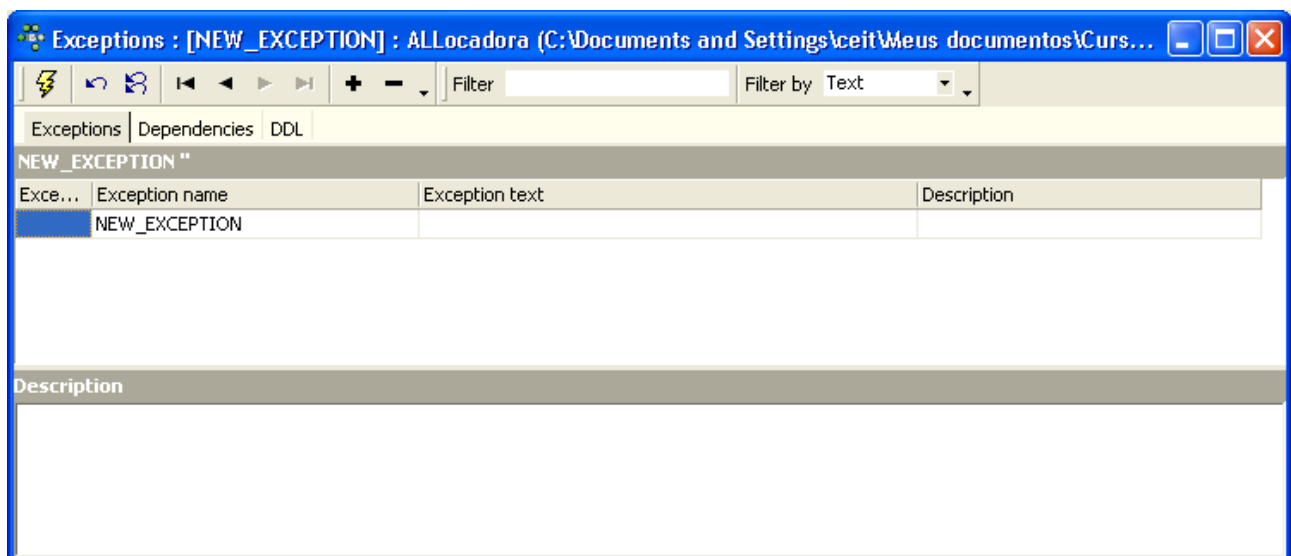


Figura 5.43a - A janela de criação de uma nova exception

4.3) Preencher os dados da *exception* solicitados.

❖ Exemplo 5.1

Por exemplo, no nosso banco “Locadora”, a tabela “Locacoes” tem um campo onde deve ser digitado o valor da locação (referente àquele filme locado). Assim, é importante validar a entrada de dados nesse campo para que seja coerente: sempre não negativo. Esse campo é o “Loca_valor”; desse modo, sempre que houver uma entrada nesse campo que não seja um valor não negativo, uma mensagem de erro deve ser emitida gerada pela *exception*, acionada pela *trigger*. Então, utilizando a janela da **Figura 5.43a**, vamos criar essa *exception* preenchendo os dados solicitados nas colunas. A instrução, neste caso, é a seguinte:

CREATE EXCEPTION Loca_nula ‘Valor da locação não pode ser nem negativa e nem nula!’

Esta instrução diz o seguinte: se o valor da locação não for maior ou igual a zero, então a *exception* “Loca_nula” será acionada pela *trigger*. A **Figura 5.43b** mostra essa *exception* criada.

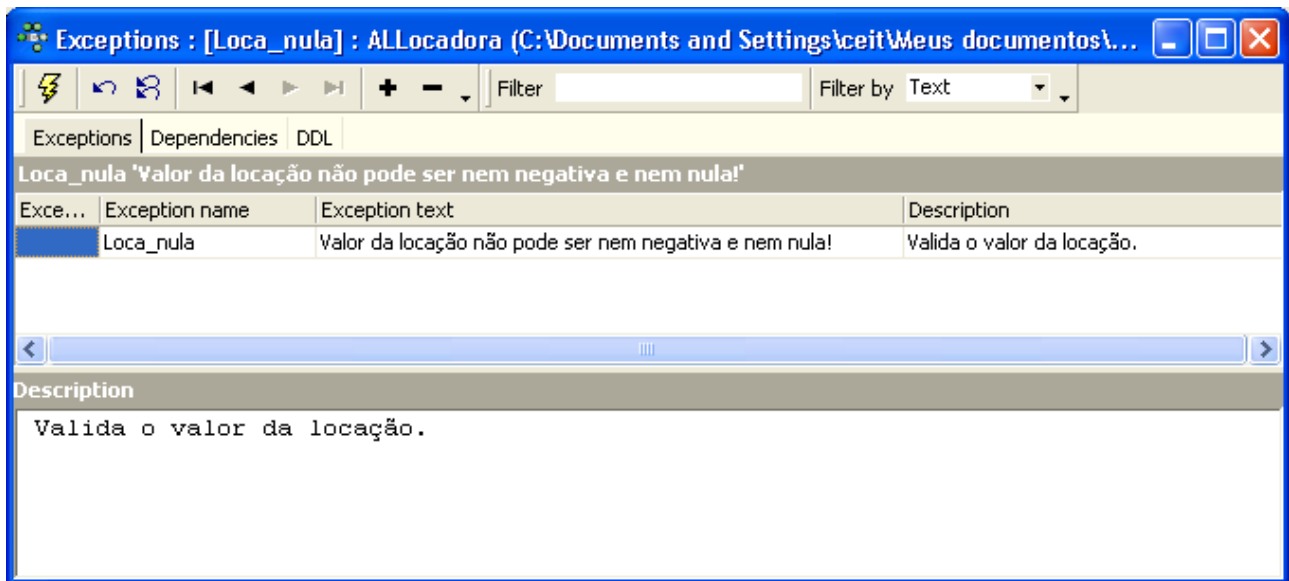



Figura 5.43b - A janela de criação de uma nova *exception*

Compile a *exception* clicando no ícone  ou pressionando teclas de atalho **Ctrl+F9**; aparece a janela da **Figura 5.43c**.

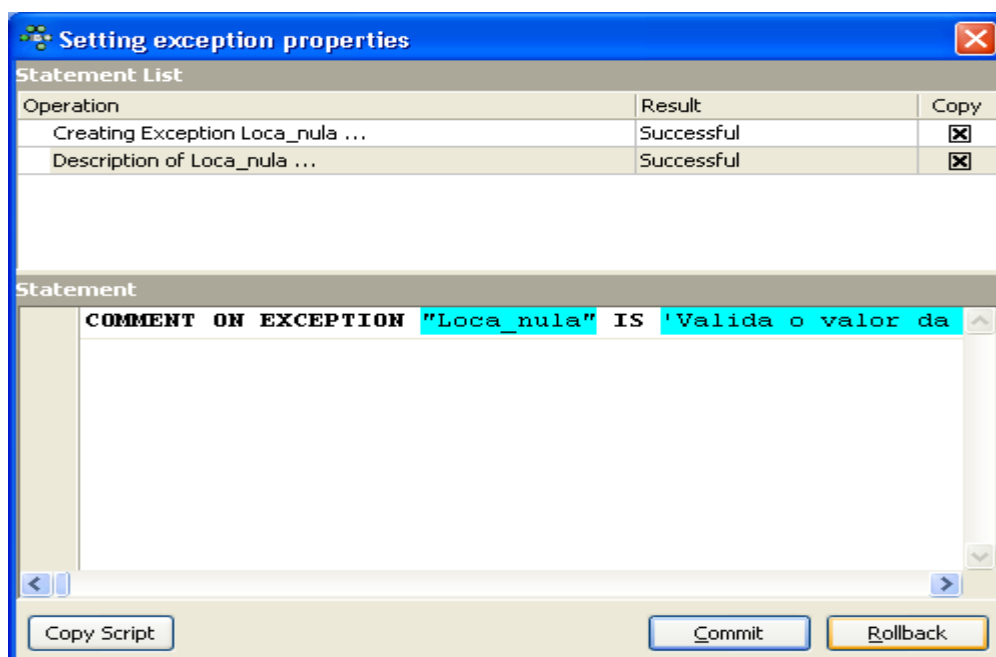


Figura 5.43c - Compilação da criação da *exception* “Loca_nula”

Agora é só *commitar* a criação da *exception* confirmando no botão **[Commit]**. Concluindo isto, a nova *exception* estará criada.

Agora que foi criada a *exception*, pode ser criada a *trigger* fazendo o seguinte:

1) Dê um clique com o botão direito do *mouse* sobre “Triggers” na relação de elementos do banco de dados e selecione a opção “New Trigger”. A janela exibida é a mostrada na **Figura 5.44a**.

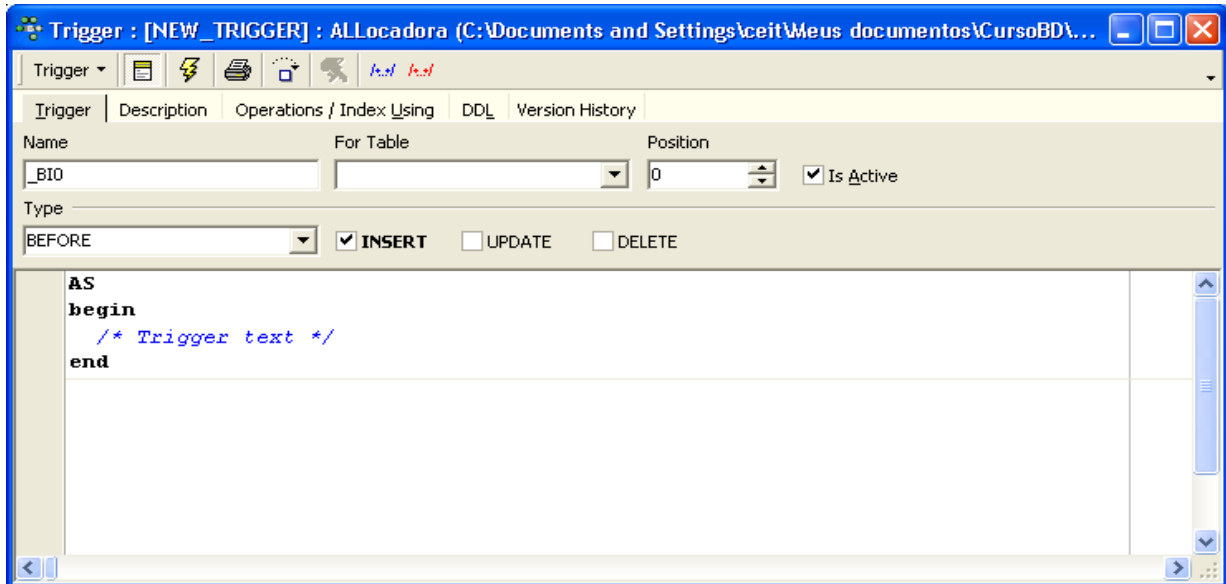


Figura 5.44a - A janela de criação de uma nova *trigger*

2) Preencha os dados solicitados na janela da **Figura 5.44a**. Observe como fica da **Figura 5.44b**.

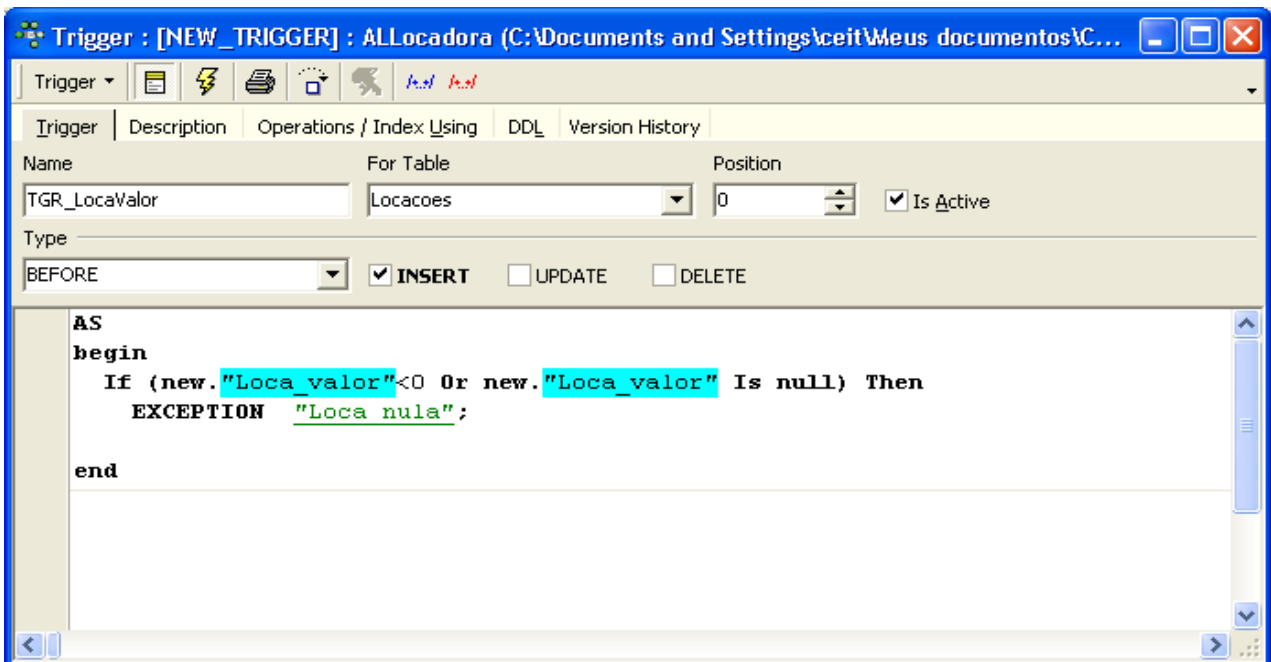



Figura 5.44b - Criação da *trigger* “Loca_nula”

Obs: O termo **new** é uma palavra chave de criação da *trigger*; ao digitar **new.** (seguido de um ponto) a relação dos campos da tabela será mostrada para que um deles seja selecionando.

3) Clique no item “Trigger” no campo superior esquerdo da janela e em seguida na opção “Compile Trigger...” para compilar; ou clique diretamente sobre o ícone de compilação , ou ainda, através de atalho, pressionando simultaneamente **Ctrl+F9**. O resultado é a janela de compilação mostrada na **Figura 5.45**.

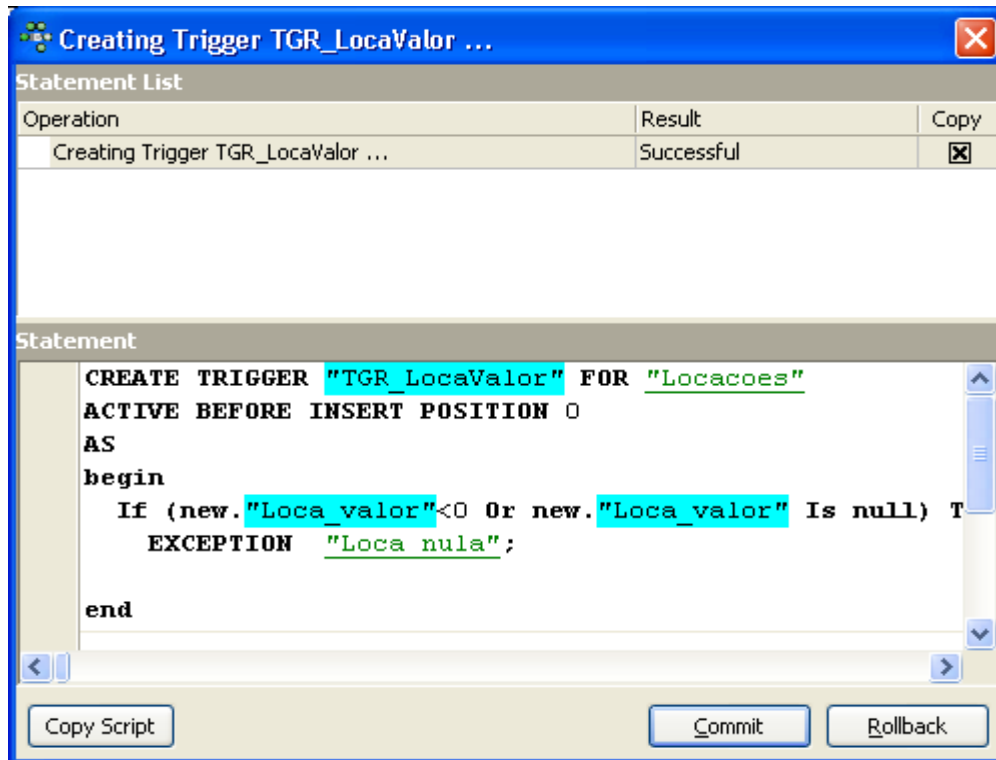


Figura 5.45 - Janela de compilação da *trigger* “Loca_nula”

4) Confirme no botão **[Commit]** para gravar a *trigger* no banco de dados.

Observe agora na **Figura 5.46** que o “DB Explorer” exibe a *trigger* e a *exception* criadas.

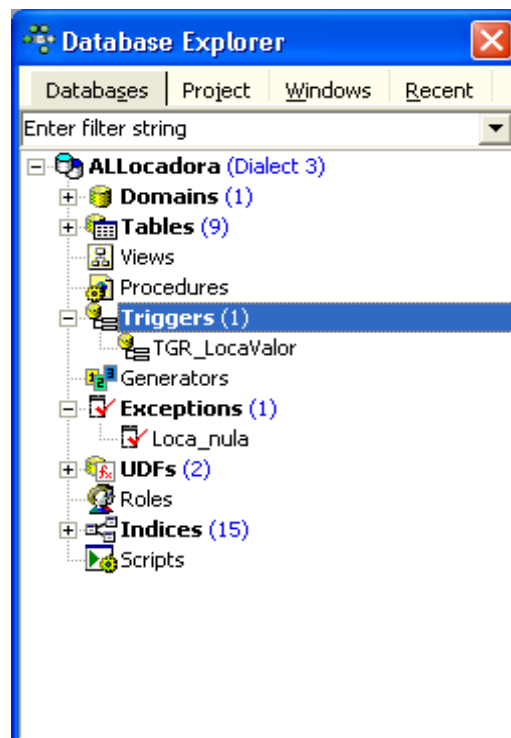


Figura 5.46 - A *exception* e a *trigger* criadas

Vamos agora testar a funcionalidade da *trigger* “TGR_LocaValor” criada anteriormente.

Dando um duplo clique na tabela “Locacoes” na lista do “DB Explorer”, e em seguida clicando na guia “Data”, vamos entrar com um registro de locação do filme “Casamento Grego” (código 292), como mostra a **Figura 5.47**.

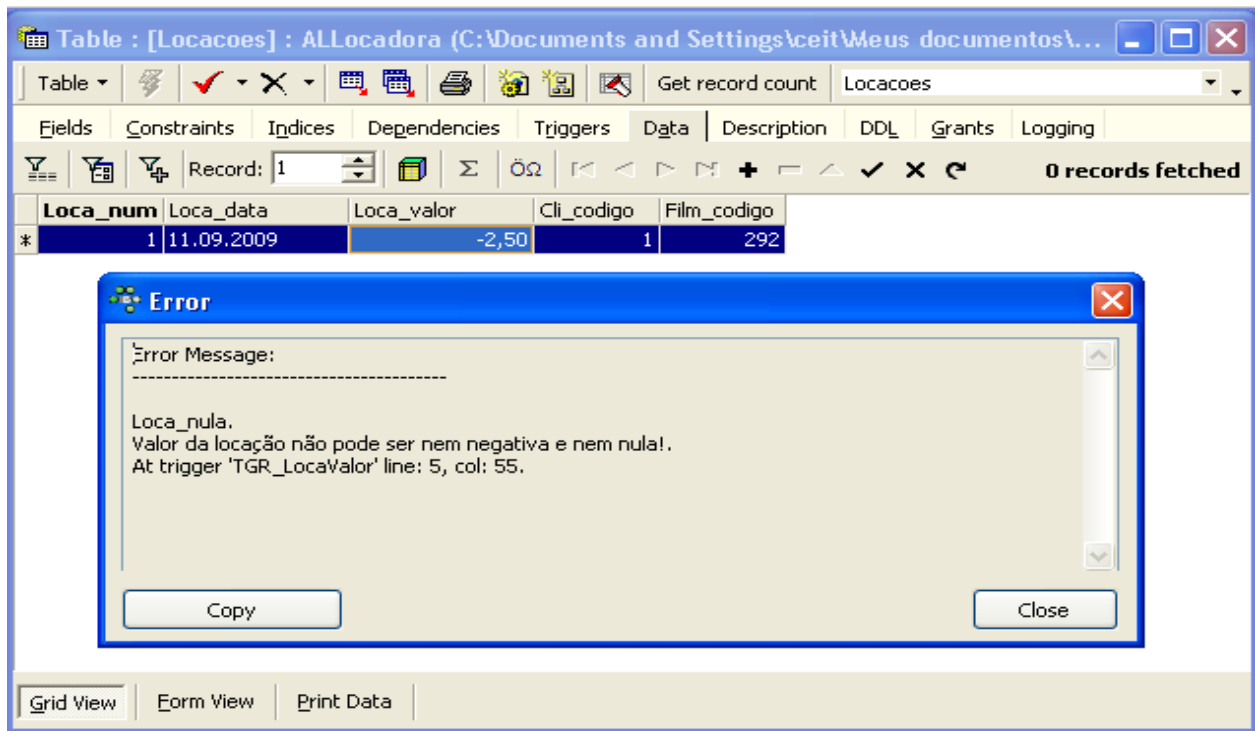


Figura 5.47 - Tentativa de locar um filme com valor negativo para a locação

Observe na **Figura 5.47** que ao tentar entrar com um valor negativo (-2,50) para a locação do filme, é gerada um erro disparado pela *trigger*, exibindo exatamente a mensagem de texto da *exception*, para impedir a entrada do dado incorreto...

5 - Criação de uma *trigger* manualmente

Também podemos criar *triggers* através de instruções sql digitadas e executadas no “SQL Editor” do IBExpert. Para isto, execute os seguintes passos:

- 1º) Na barra de *menus* clique em **Toos/SQL Editor** ou pressione direto a tecla de função **F12**.
- 2º) Aparecendo o editor, digite as instruções como mostra a **Figura 5.48**.

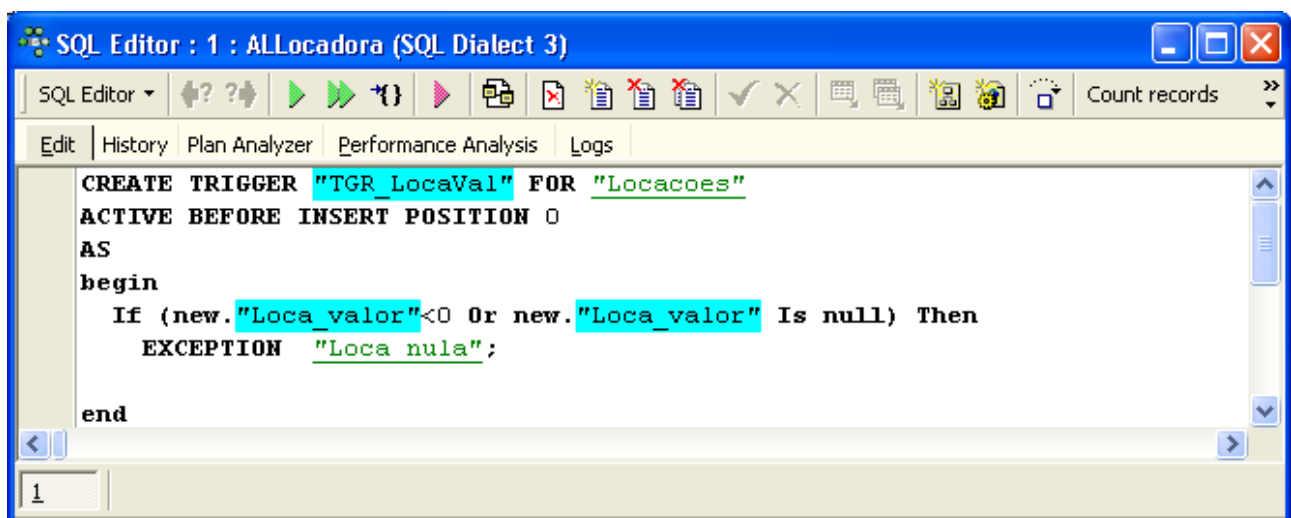



Figura 5.48 - Criando uma *trigger* no “SQL Editor do IBExpert

Observe que alteramos o nome da *trigger* para “TGR_LocaVal”, pois já existia uma com o nome “TGR_LocaValor”.

3º) Crie a *trigger* executando as instruções com **F9** (ou clique diretamente no ícone )

Nota: A criação manual através de instruções sql no editor de um gerenciador de bancos de dados deve ser feita com bastante critério e pesquisa. Dependendo do gerenciador, alguns termos extras devem ser digitados para que dê certo. Por exemplo, para o gerenciador nativo do Interbase - IBConsole - as instruções para este nosso exemplo seriam as seguintes:

```
SET TERM ^ ;
CREATE TRIGGER "TGR_LocaVal" FOR Locacoes
ACTIVE BEFORE INSERT POSITION 0
AS
begin
If (new."Loca_valor"<0 Or new."Loca_valor" Is null) Then
    EXCEPTION "Loca_Nula";
End ^
SET TERM ; ^
```

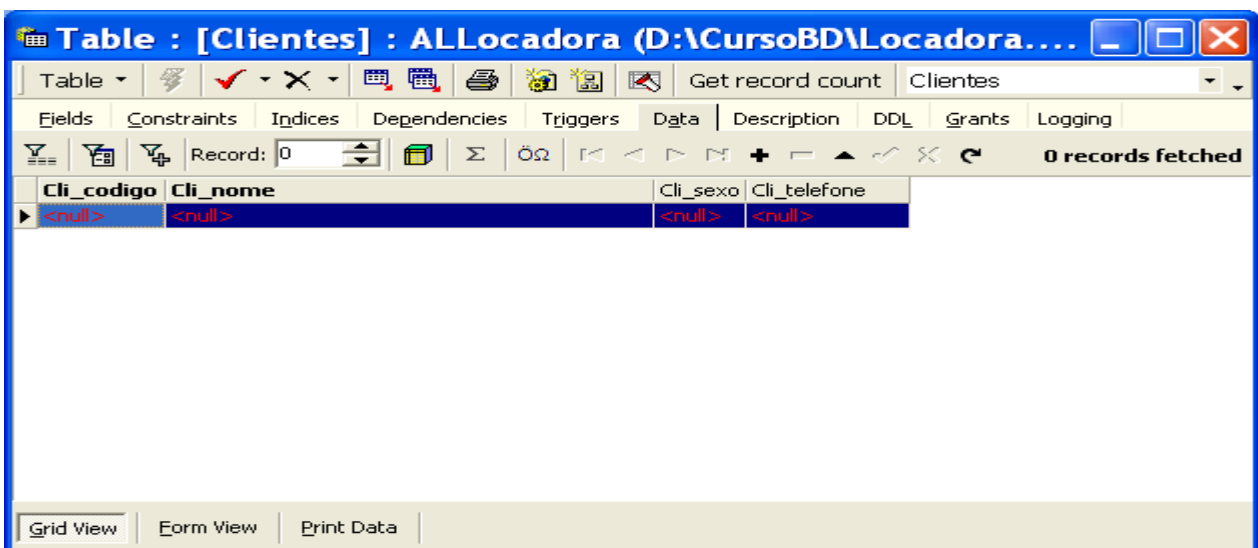
O termo **SET TERM** é usada para informar ao servidor que o caracter terminador da instrução é o símbolo “^” e não mais o tradicional do padrão SQL “;”. Assim, o ISQL (Interpretador SQL) faz uma pré-análise da instrução e a envia diretamente ao servidor como um comando único. Isto é importante, pois caso contrário ocorreria erro na criação da *trigger*, pois o ISQL interpretaria, normalmente, o ponto e vírgula (;) como o terminador de uma instrução.

Bem entendido, isto vale para IBConsole. No IBExpert, apesar de sua compilação gerar automaticamente esse termo, ao digitá-lo manualmente no editor pode dar problema na execução; portanto muito cuidado ao tentar criar *triggers* manualmente com instruções sql.

5.9 - Entrando com dados nas tabelas

Na prática, os dados devem ser gravados nas tabelas através da aplicação cliente (uma aplicação escrita em linguagem de alto nível), mas como estamos estudando bancos de dados, podemos mostrar como isto pode ser feito manualmente.

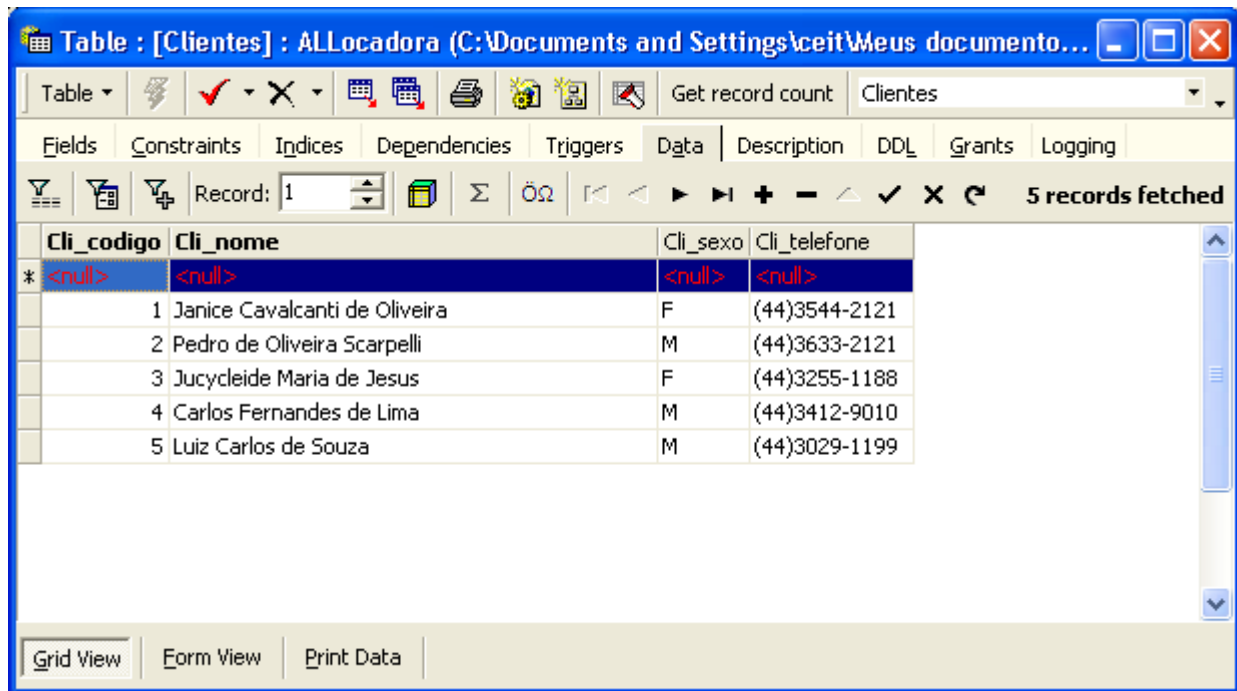
1) Faça o *login* do servidor local e expanda a árvore de objetos do banco cujo *alias* é ALLocadora, e em seguida dê um duplo clique sobre a tabela “Clientes”; depois selecione a guia “Data”. A janela que se apresenta é a da **Figura 5.49a** exibindo os campos da tabela em estilo “*folha de dados*”, pronta para receber os dados de um cliente.



Cli_codigo	Cli_nome	Cli_sexo	Cli_telefone
<null>	<null>	<null>	<null>

Figura 5.49a - A tabela “Clientes” em modo “*folha de dados*” pronta para receber registros

2) Digite os dados nos campos indicados, observando que o campo chave primária (Cli_codigo) e o campo "Cli_nome" (chave secundária) devem ser obrigatoriamente preenchidos com valores compatíveis, pois foram definidos como **Not Null** (lembra?!). Para incluir um novo registro basta clicar na tecla **+** da barra de ferramentas. Veja como ficam os primeiros cinco clientes cadastrados na **Figura 5.49b**.



Cli_codigo	Cli_nome	Cli_sexo	Cli_telefone
* <null>	<null>	<null>	<null>
1	Janice Cavalcanti de Oliveira	F	(44)3544-2121
2	Pedro de Oliveira Scarpelli	M	(44)3633-2121
3	Jucycleide Maria de Jesus	F	(44)3255-1188
4	Carlos Fernandes de Lima	M	(44)3412-9010
5	Luiz Carlos de Souza	M	(44)3029-1199

Figura 5.49b - Entrando com registros na tabela "Clientes"

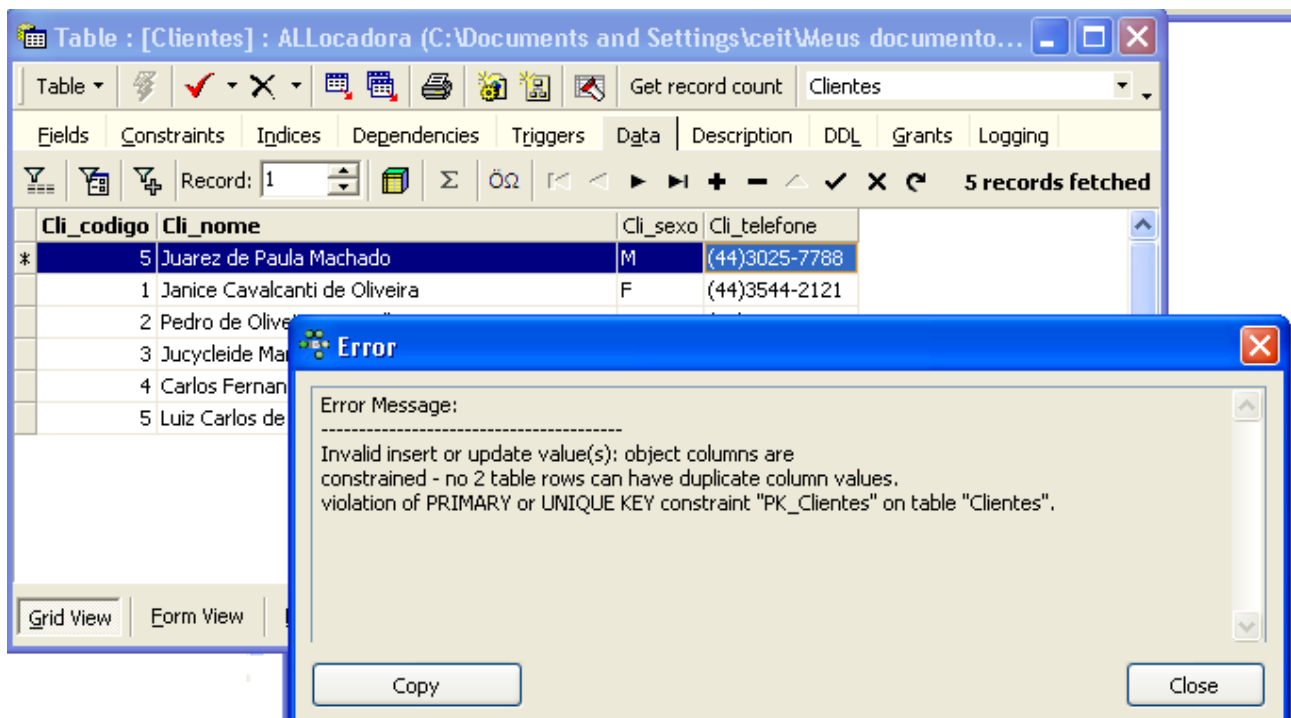



Figura 5.49c - Tentando inserir um registro com chave primária duplicada

Observe a **Figura 5.49c**: o que aconteceu? Houve uma tentativa de inserir um novo cliente com código 5, mas já existia um cliente com esse código. E como o campo "Cli_codigo" é chave primária da tabela, o sistema rejeitou essa entrada mantendo sua integridade, e gerando uma mensagem de erro...

Para confirmar todas as entradas (cadastros) na tabela, deve ser clicado o ícone  ou acionar o menu **Tabela/Commit Transaciton** para *commitar* (gravar efetivamente os dados no banco). O resultado é a exibição da janelinha “Confirmation” mostrada na **Figura 5.49d**. Confirme no botão **[Yes]**. Pronto, agora todos os clientes estão devidamente cadastrados na tabela...

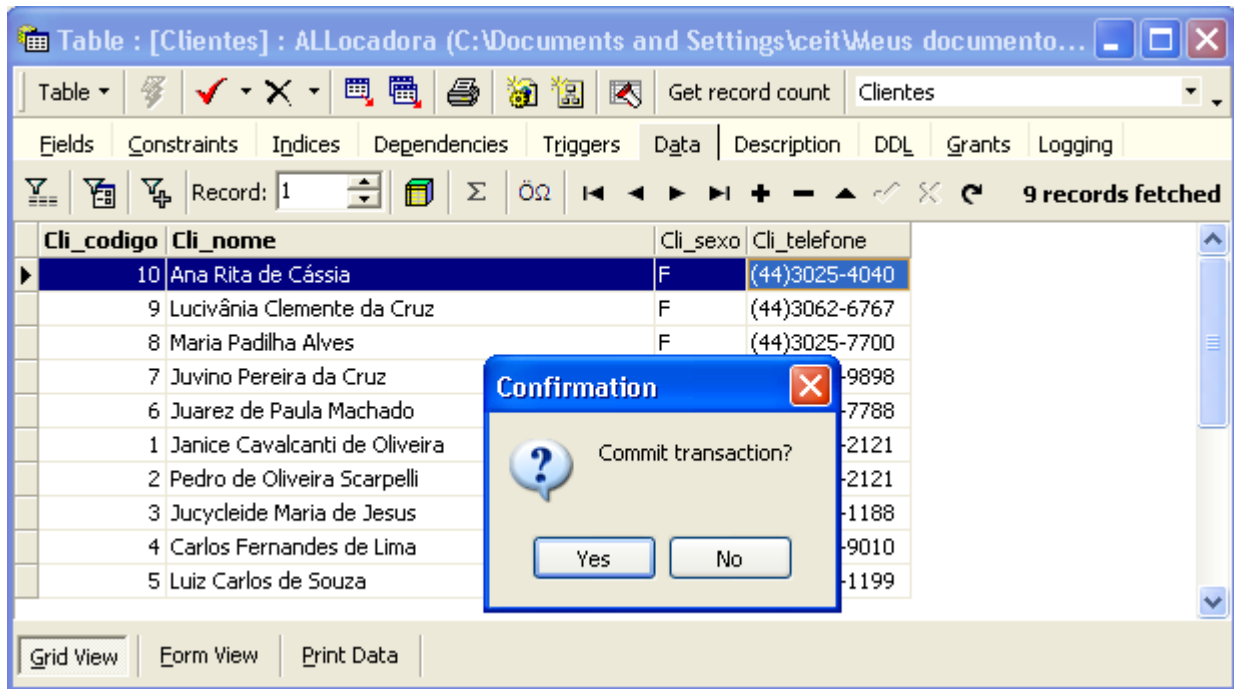


Figura 5.49d - Commitando a tabela “Clientes” para efetivar a gravação dos dados

Repita o processo e cadastre outros alunos.

Após dado o *commit*, a tabela “Clientes” agora se apresenta com seus registros ordenados pelo “Cli_codigo”, como mostra a **Figura 5.49e**. Você poderia explicar por que isso aconteceu?!

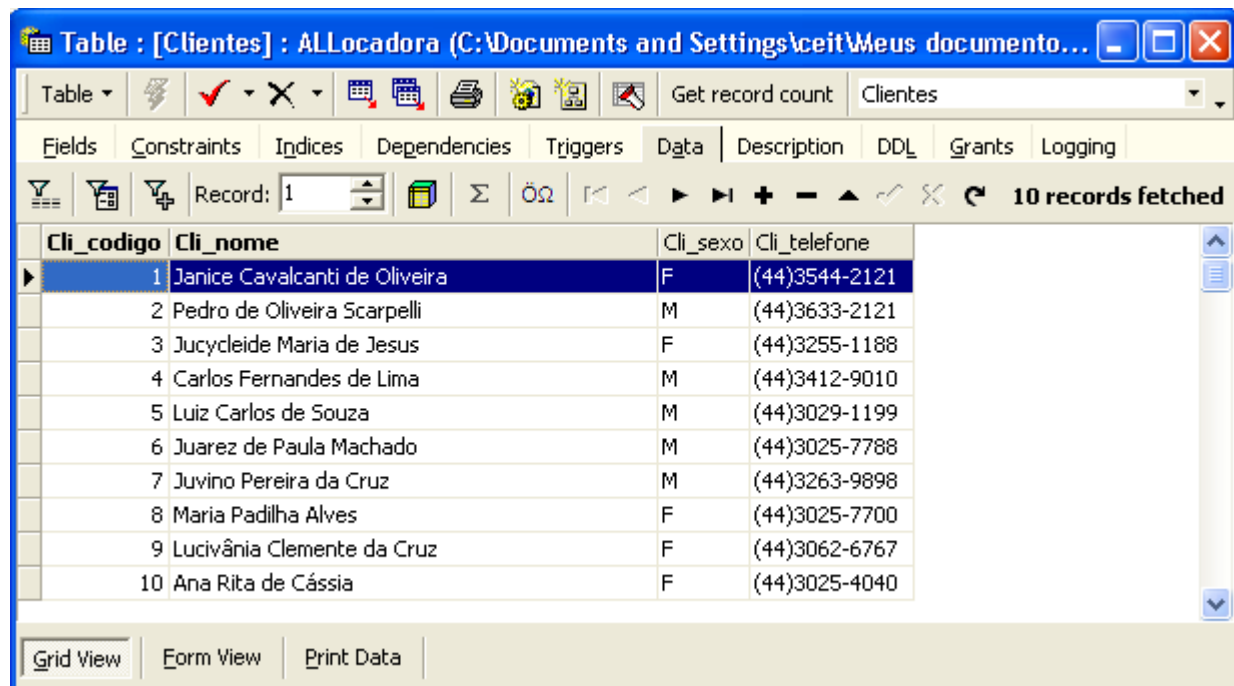


Figura 5.49e - A “Clientes” com seus registros reordenados pela chave primária

A entrada de dados em uma tabela deve obedecer aos critérios definidos no MER, verificando as dependências desta com outras entidades (tabelas). Neste caso, a entrada de dados na tabela “Locacoes”, por exemplo, só deve ser feita depois dos dados digitados nas tabelas “Clientes” e “Filmes”. Mas, por outro lado, os filmes só poderão ser cadastrados se antes forem cadastrados os gêneros, os tipos de filmes e os diretores; e assim por diante... Por isso, ao cadastrar entidades numa tabela é preciso verificar antes se existe alguma dependência a ser considerada. Como cadastrar um novo filme se não existe um diretor, um gênero, e um tipo para esse filme?!!. O MER para o criador do banco de dados é como se fosse a planta de uma casa para o mestre de obras!

A **Figura 5.50** mostra a barras de ferramentas utilizada para entrada de dados nas tabelas; no caso, destacando a tabela “Clientes”...

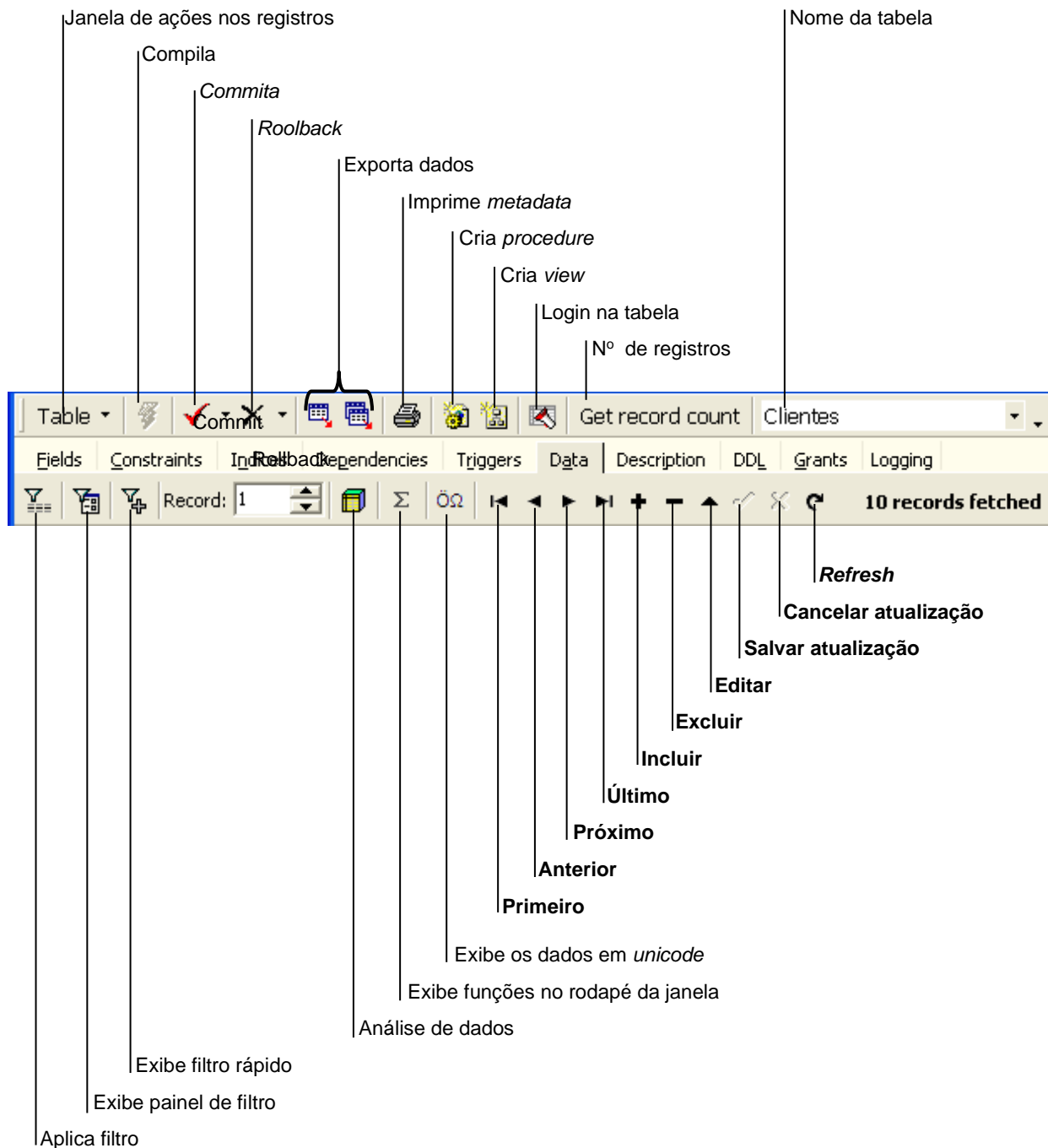


Figura 5.50 - Ferramentas para manipular registros de uma tabela

5.10 - Tipos de dados suportados pelo Firebird

O Firebird suporta muitos tipos de dados; mais ou menos tipos que os outros bancos corporativos. A **Tabela 5.1** mostra esses tipos de dados, que define automaticamente um domínio: a faixa de valores permitida e as operações que poderão ser executadas com os dados daquele tipo.

Tipo	Descrição
BIGINT	Armazena valores numéricos com sinal na faixa de -9223372036854775808 a +9223372036854775807. A faixa de valores sem sinal vai de 0 até ... 18446744073709551615.
BLOB	Armazena figuras e textos dentro de uma sequência de <i>bytes</i> . Para um SUB_TYPE TEXT o tamanho máximo é 65535 (64Kb). Serve para armazenar grandes textos (<i>memo</i>), imagens, fotos, ícones, etc.
CHAR(n)	Armazena caracteres com tamanho fixo n de no máximo 32767 se o campo não for um índice; caso seja um índice o valor máximo para n é 252.
DATE	Armazena datas no intervalo desde 01/01/0001 até 31/12/9999.
DECIMAL(n,d)	Armazena valores numéricos reais de tamanho n com d casas decimais e suporta valor máximo igual ao tipo <i>double precision</i> .
DOUBLE PRECISION	Armazena valores numéricos reais tal como o tipo Float, porém em 64 bits, na faixa de 2.22507E-308 e 1.79769E+308.
FLOAT	Armazena valores numéricos reais em ponto flutuante de 32 <i>bits</i> na faixa de 1.175E-38 até 3.402E38 (com precisão simples).
INTEGER	Armazena valores numéricos inteiros de -2147483648 a +2147483647.
NUMERIC(n,d)	Armazena valores numéricos reais com formatação do tamanho n e casas decimais d . Teoricamente, o valor máximo desse é igual ao valor máximo suportado por <i>double precision</i> .
SMALLINT	Armazena valores inteiros de -32768 a +32767.
TIME	Armazena horas, minutos e segundos no formato de horas, desde 00:00:00:00 até 23:59:99:99.
TIMESTAMP	Armazena data e hora simultaneamente, armazenando valores de data (Date) e horas (Time), juntos.
VARCHAR(n)	Armazena caracteres em campo com tamanho variável máximo de n (até 32767 se campo não for um índice).

Tabela 5.1 - Tipos de dados suportados pelo Firebird

Por exemplo, para criar uma tabela “Empregado” e colocar uma foto num campo tipo BLOB

```
CREATE TABLE Empregado
(
  Emp_mat    INTEGER NOT NULL PRIMARY KEY,
  Emp_nome   VARCHAR(50) NOT NULL,
  Emp_foto   BLOB SUB_TYPE 0,
  Emp_exp    BLOB SUB_TYPE 1 SEGMENT SIZE 80
);
```

Tamanho fixo de cada bloco gravado

No código acima, o parâmetro **0** (zero) de SUB_TYPE indica que o dado é uma imagem; o parâmetro **1** é para texto. Veja o **item 5.15** mais adiante.

5.11 - Trabalhando com Generators

Generators são elementos que criam sequências numéricas. Sendo empregados em conjunto com *trigger* e/ou *stored procedure*, são utilizados para simular um campo auto-incremento como nos bancos do tipo *desktop*. Uma outra utilização de *generator* é para evitar chaves duplicadas em campos numéricos. O valor *default* é 0 (zero), mas é atualizado toda vez que é chamada a função *Gen_ID()*, mas também pode ser empregado para decrementar valores em um campo. É possível saber qual o valor atual do generator: basta passar o valor 0 (zero) para a função *Gen_ID()*. Observe as instruções abaixo:

```
CREATE GENERATOR <Nome_do_Generator>;      //Cria um generator
SET GENERATOR    < Nome_do_Generator> TO <value>; //Atualiza o generator
DROP GENERATOR   < Nome_do_Generator>;      //Deleta o generator
```

❖ Exemplo 5.2

Criar um *generator* para que o campo **Func_codigo** da tabela “Funcionario” fique auto-incrementável (é claro que a tabela “Funcionarios” já deve existir no banco de dados). A **Figura 5.51** mostra nosso banco de dados com essa tabela criada do mesmo modo que foram criadas as outras. Os campos são:

- **Func_codigo** Código do funcionário (auto-incrementável pelo *generator*);
- **Func_sexo** Sexo do funcionário (“M” “m”, “F” ou “f” - validado com o domínio *Dom_sexo*);
- **Func_nome** Nome do funcionário;
- **Func_fone** Telefone do funcionário.
-

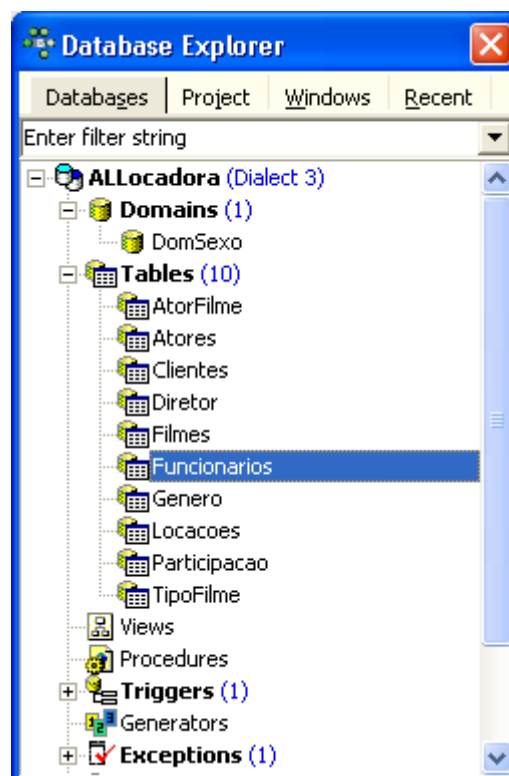



Figura 5.51 - O banco de dados com a nova tabela “Funcionários”

Junto com o *generator* vamos criar, também, uma *trigger* que chame esse *generator* para executar a tarefa de incrementar o campo “Func_codigo” toda vez que um novo registro for criado.

Mas, atenção: primeiro deve ser criado o *generator* e depois a *trigger* (nesta ordem), conforme mostram as instruções a seguir...

```
CREATE GENERATOR Gen_Func;
```

```
CREATE TRIGGER "TGR_AutoIncrementa" FOR "Funcionarios" BEFORE
INSERT POSITION 0
AS
BEGIN
    New."Func_codigo" = NEXT VALUE FOR Gen_Func;
END
```

As figuras 5.52a e 5.52b mostram as sequências de criação e *commitação* da instrução sql para criar o generator “Gen_Func”. Na **Figura 5.52a** o código de criação do generator é digitado dentro do “SQL Editor”; e para executar a sql basta clicar no ícone .

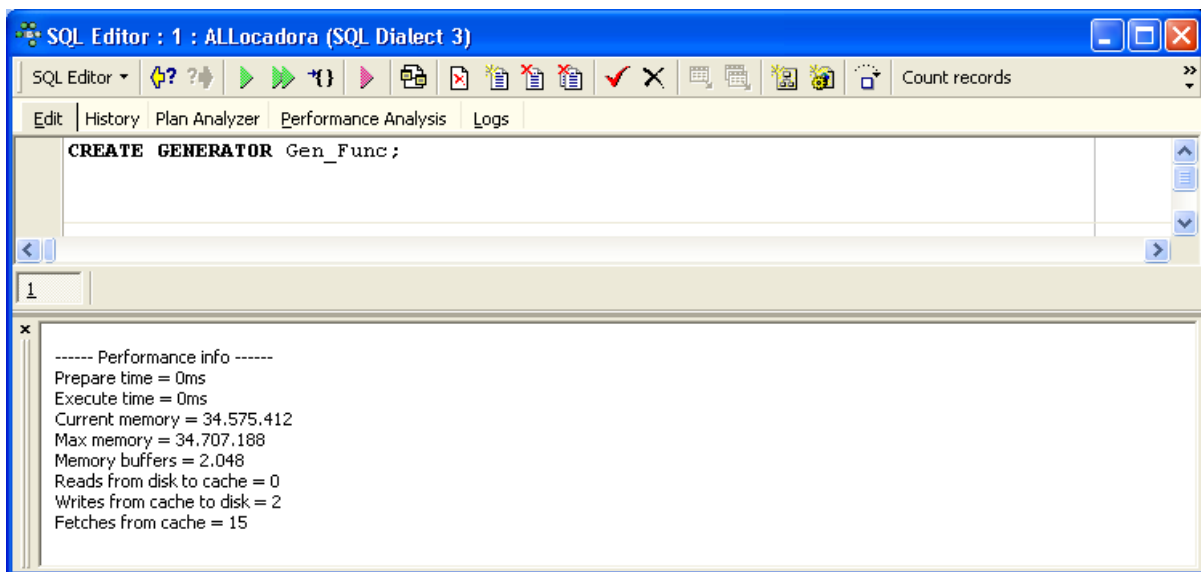



Figura 5.52a - Executando a instrução sql para criar o generator

Depois de executada a instrução, esta deve ser *commitada* clicando no ícone . O resultado é exibido na **Figura 5.52b**, mostrando que a transação foi executada corretamente.

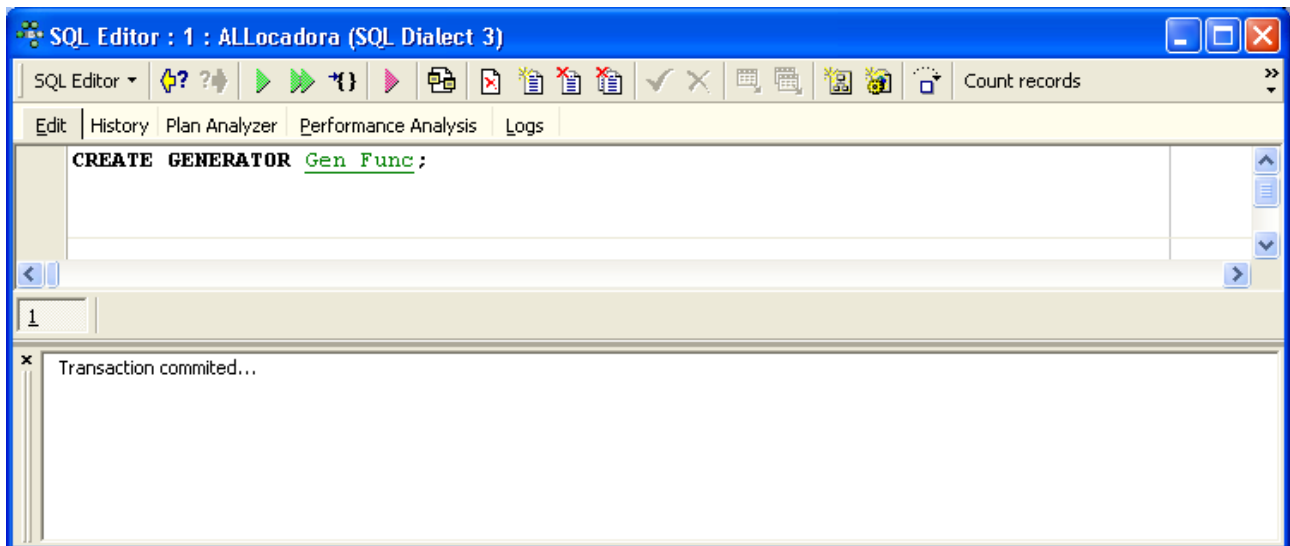


Figura 5.52b - Commitando a transação de criação do generator

Para que o generator possa ser utilizado, é necessário indicar isto por ocasião da criação do campo na tabela. Observe a **Figura 5.53**: ao definir o campo "Func_codigo", na coluna "AutoInc" dê um clique no quadradinho para indicar na janela "Autoincrement Field" que será usado o *generator existente* (no caso o "Gen_Func" criado).

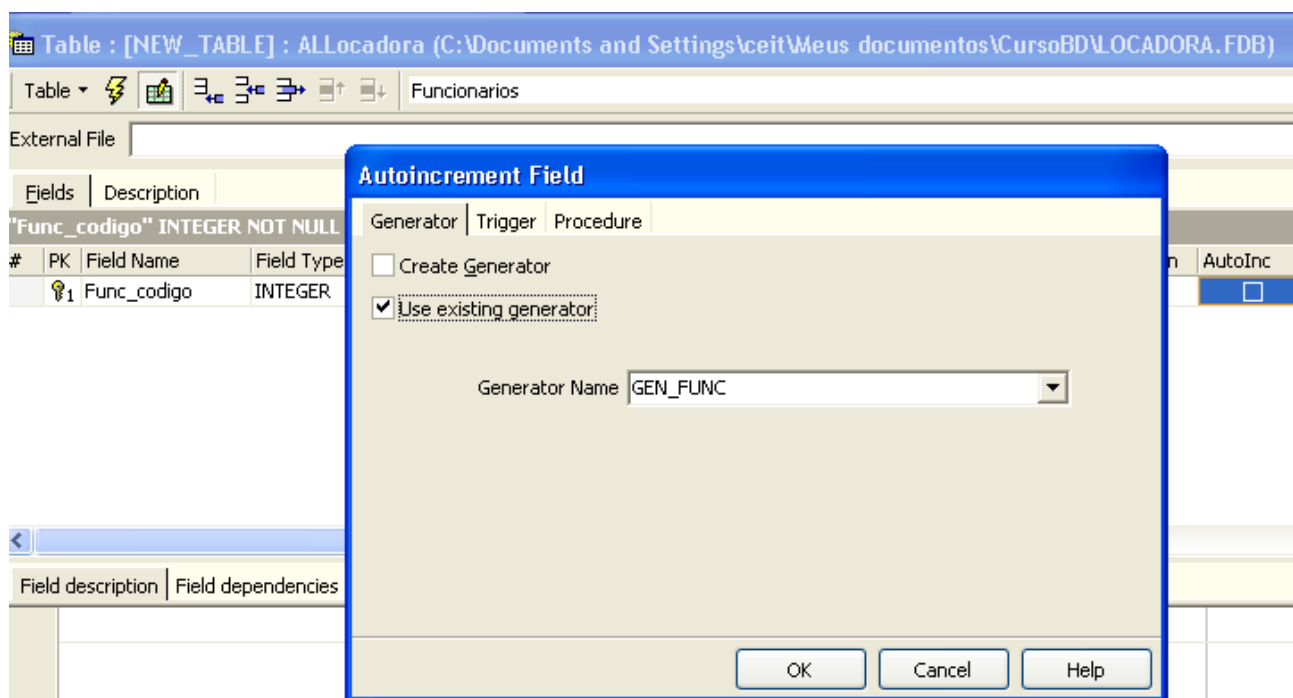


Figura 5.53 - Definindo o generator como elemento incrementador do campo "Func_codigo"

Confirme no botão [OK] e crie os outros campos...

A **Figura 5.54** mostra o banco de dados exibindo a nova tabela "Funcionarios" e o *generator*.

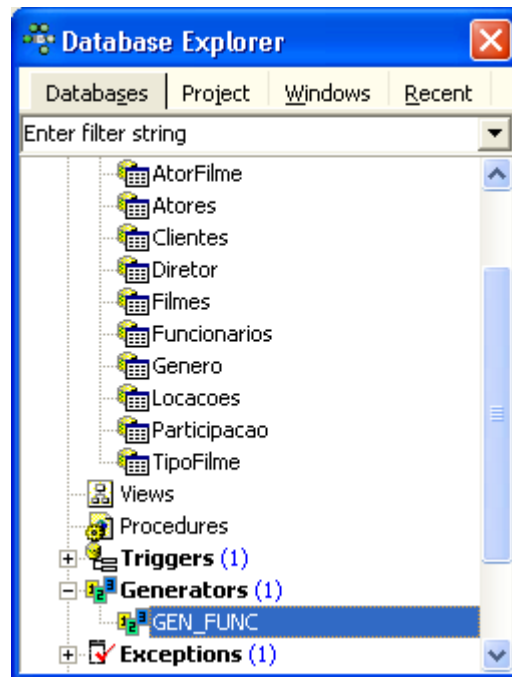


Figura 5.54 - A nova tabela “Funcionarios” e o generator

Para criar a *trigger* carregamos o “SQL Editor” do IBExpert e digitamos o código com as instruções sql, conforme mostrado na **Figura 5.55**.

Atenção: Ao digitar (manualmente) o código de criação da *trigger* o nome do campo deve ser colocado entre aspas (“ ”), caso contrário o compilador de sql informa que o “campo não pertence à tabela”. Isto porque o banco “Locadora” foi criado com o “CharSet” NONE.

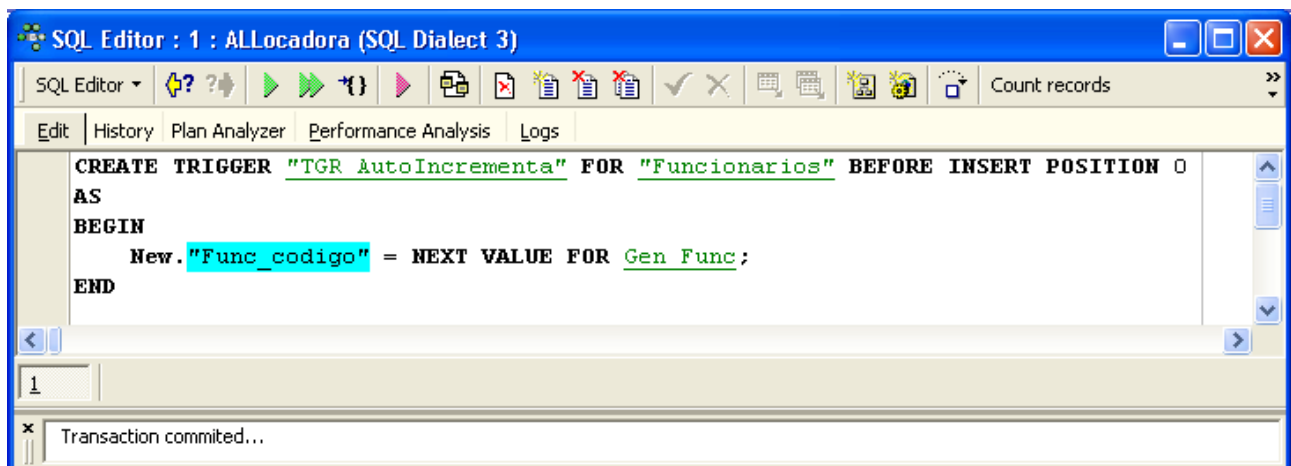


Figura 5.55 - A *trigger* criada

A **Figura 5.56** mostra a nova *trigger* (TRG_AutoIncrementa) criada no banco de dados.

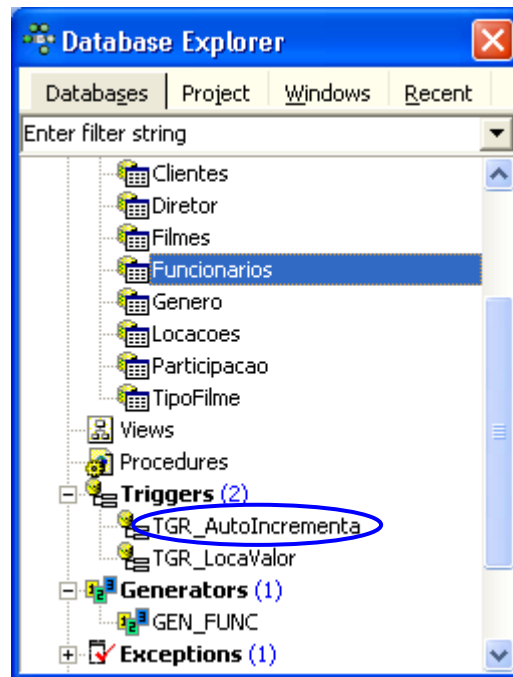



Figura 5.56 - A nova *trigger* criada

Agora vamos testar nosso *generator* criado anteriormente. Com um duplo clique na tabela “Funcionários”, ela se apresenta no modo “*folha de dados*”, pronta para ser preenchida com os registros dos funcionários. Observe na **Figura 5.57a**: depois de entrar com dois registros, estamos preparando a entrada de um terceiro. Ao terminar de entrar com os três campos (deixando o campo “Func_codigo”) e clicando no botão  para atualizar, veja o que acontece na janela na **Figura 5.57b**.

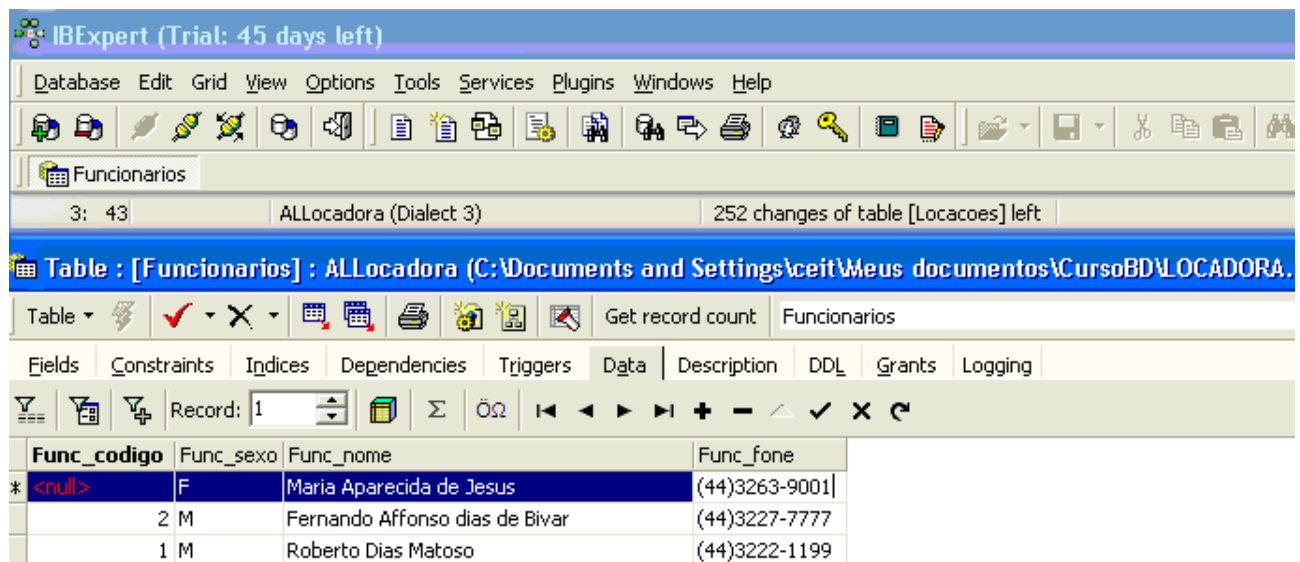


Figura 5.57a - Entrando com dados na tabela “Funcionarios” (1)

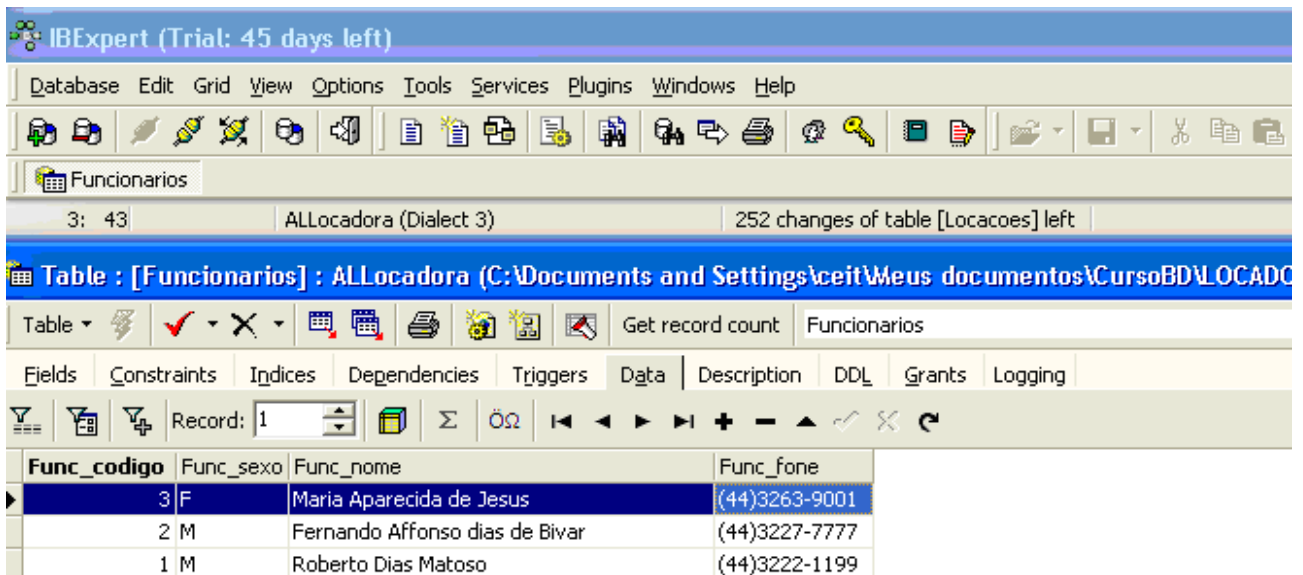


Figura 5.57b - Entrando com dados na tabela “Funcionarios” (2)

Observe na **Figura 5.57b** que devido à ação do generator “Gen_Func”, o código da funcionária “Maria Aparecida de Jesus” foi criado automaticamente como 3 (sequência do código anterior 2).

5.12 - Views (visões)

Na terminologia dos bancos de dados, uma “visão” é uma “tabela” única derivada de outra tabela, podendo ser uma tabela básica ou uma visão previamente definida. Entretanto, é importante frisar que uma visão não existe de forma física, apenas virtualmente; ela não é armazenada no banco de dados. Por isso, as operações com visões são extremamente limitadas, apesar de não existir um limite para as consultas. São definidos apenas dois tipos de visões:

- Visões apenas para leitura – quando os dados não podem ser editados;
- Visões de atualização – quando é possível alterar o estado do banco de dados.

Na prática, as visões são utilizadas para realizar tarefas simples como:

- ✓ Restringir o acesso dos usuários;
- ✓ Mostrar apenas determinadas algumas colunas;
- ✓ Filtragem de dados já previamente formatados.

Resumindo, é importante observar que:

- Uma visão de uma única tabela de definição é atualizável se a visão contiver entre seus atributos a chave primária da relação básica, bem como todos os atributos com restrição NOT NULL que não contiverem valores *default* especificados.
- As visões definidas a partir de várias tabelas utilizando junções, em geral não são atualizáveis.
- As visões definidas usando-se as funções de agrupamento e agregadas não são atualizáveis.

Conclusão: Embora possam ser empregadas para “substituir” tabelas em algum momento, as visões são bastante limitadas e sua utilização na prática não é muito usual.

❖ Exemplo 5.3

Criar uma visão que mostre os nomes dos diretores com os nomes de seus respectivos filmes.

```
CREATE VIEW DirFilme
AS SELECT D."Dir_nome", F."Film_titorig"
FROM "Diretor" D, "Filmes" F
WHERE D."Dir_codigo" = F."Dir_codigo"
```

Onde **D** e **F** são os *aliases* (apelidos) das tabelas “Diretor” e “Filmes”, respectivamente.

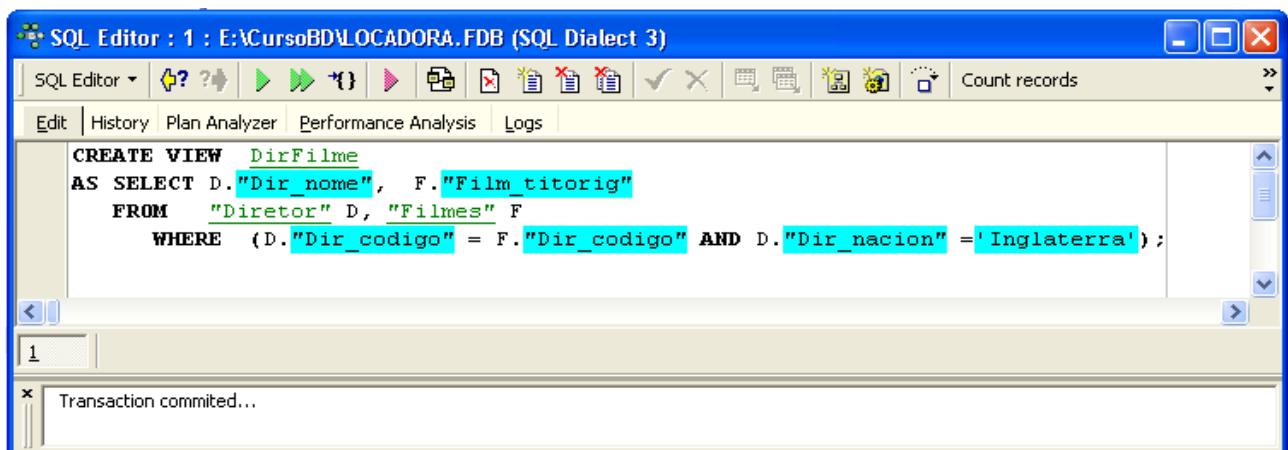


Figura 5.58 - Criação/compilação de uma visão no “SQL Editor do IBExpert”

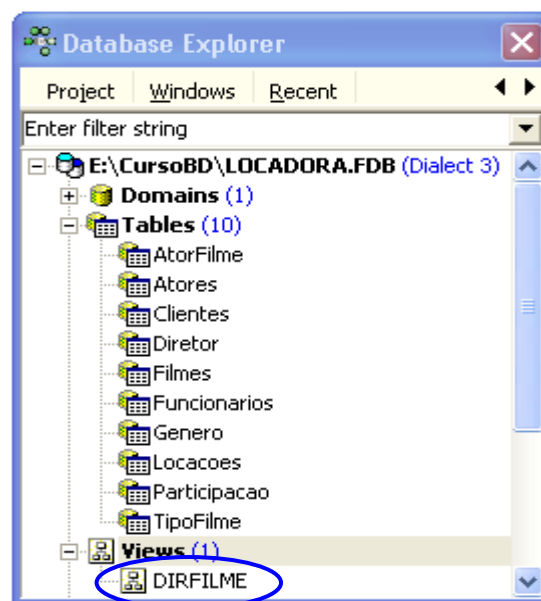


Figura 5.59 - A visão exibida no “DB Explorer”)

Dir_nome	Film_titorig
Alfred Hitchcock	Vertigo
Ridley Scott	Black Hawk Down
Charles Chaplin	City Lights
Charles Chaplin	Modern Times
David Lynch	Blue Velvet
Tony Scott	Deja vu
David Lean	Lawrence of Arabia
David Lean	The Bridge on the River Kwai
Roland Joffé	The Mission
James Ivory	A Room with a View
James Ivory	The Color of Money
Terence Young	Dr. No
Terence Young	From Russia With Love
Terence Young	From Russia With Love
Terence Young	Thunderball
Terence Young	You Only Live Twice
Irvin Kershner	Never Say Never Again
Peter R. Hunt	On Her Majesty's Secret Service
Lewis Gilbert	The Spy Who Loved Me
Lewis Gilbert	Moonraker
John Glen	For Your Eyes Only
John Glen	Octopussy
John Glen	A View to a Kill
John Glen	The Living Daylights
John Glen	Licence to Kill

Figura 5.60 - Visão dos diretores ingleses com seus respectivos filmes

A **Figura 5.60** mostra a listagem dos diretores ingleses com seus respectivos filmes, obtida através da visão “DirFilme” criada com o *script* mostrado na **Figura 5.58**.

5.13 - Trabalhando com Stored Procedures

Stored procedures são rotinas que podem ser armazenadas no banco de dados e executadas através de aplicações que o acessam. É uma coleção de instruções sql que ajudam no gerenciamento do banco de dados e executando tarefas repetitivas, reduzindo o tráfego na rede. Isto dá uma sensível melhoria na *performance* do gerenciamento dos dados, além de permitir criar dispositivos de segurança. Esses procedimentos armazenados também aceitam parâmetros, o que aumenta a sua flexibilidade. A instrução sql para criar uma *stored procedure* em bancos Interbase/Firebird é a descrita a seguir.

```
CREATE NomeProcedure ([param1 Tipo, param2 Tipo, ...])
[RETURNS valor Tipo]
AS
BEGIN
    <Corpo da procedure>
    SUSPEND;
END
```


param1, param2,... são os parâmetros de entrada, usados para modificar o comportamento da *stored procedure*, fazendo com eles sejam utilizados por ela no processamento.

Returns dá o valor retornado (resultado obtido pela *stored procedure*); é o parâmetro de saída.

SUSPEND causa uma pausa temporária na execução da *stored procedure* até que o programa busque os valores dos parâmetros de saída.

A sintaxe para criar uma *stored procedure* apresentada anteriormente é simples; entretanto, podem ocorrer casos em que essa sintaxe torna-se muito complexa com a geração de erros irritantes e às vezes difíceis de serem detectados. Por isso, o IBExpert (nosso anjo da guarda) nos guia com segurança através desses percalços que podem aparecer na criação do procedimento.

Obs) Os parâmetros podem ter qualquer tipo de dado, exceto BLOB ou ARRAY

Uma *stored procedure* admite comentários, desde que colocados entre */** e **/*; como por exemplo, */* Este é um comentário..*/*

❖ Exemplo 5.4

Vamos criar uma *stored procedure* bem simples: contar o número de clientes que existe atualmente na tabela “Clientes”. O código geral dessa *stored procedure* é mostrado a seguir...

```
CREATE PROCEDURE ContaClientes
RETURNS (QuantCli INTEGER)
AS
BEGIN
    SELECT COUNT (*) FROM "Clientes" INTO : "QuantCli";
    SUSPEND;
END
```

O código acima pode ser colado diretamente no “SQL Editor” do IBExpert, observando que o nome da tabela está entre aspas (“ ”) devido ao Charset **NONE** do banco na sua criação, caso contrário será gerado um erro na compilação, informando que a tabela não existe (!).

Para criar a mesma *stored procedure* de maneira interativa com o IBExpert faça o seguinte:

1º) Clique com o botão direito do *mouse* sobre “Procedures” no “DB Explorer”; a janela mostrada é a da **Figura 5.61a**.

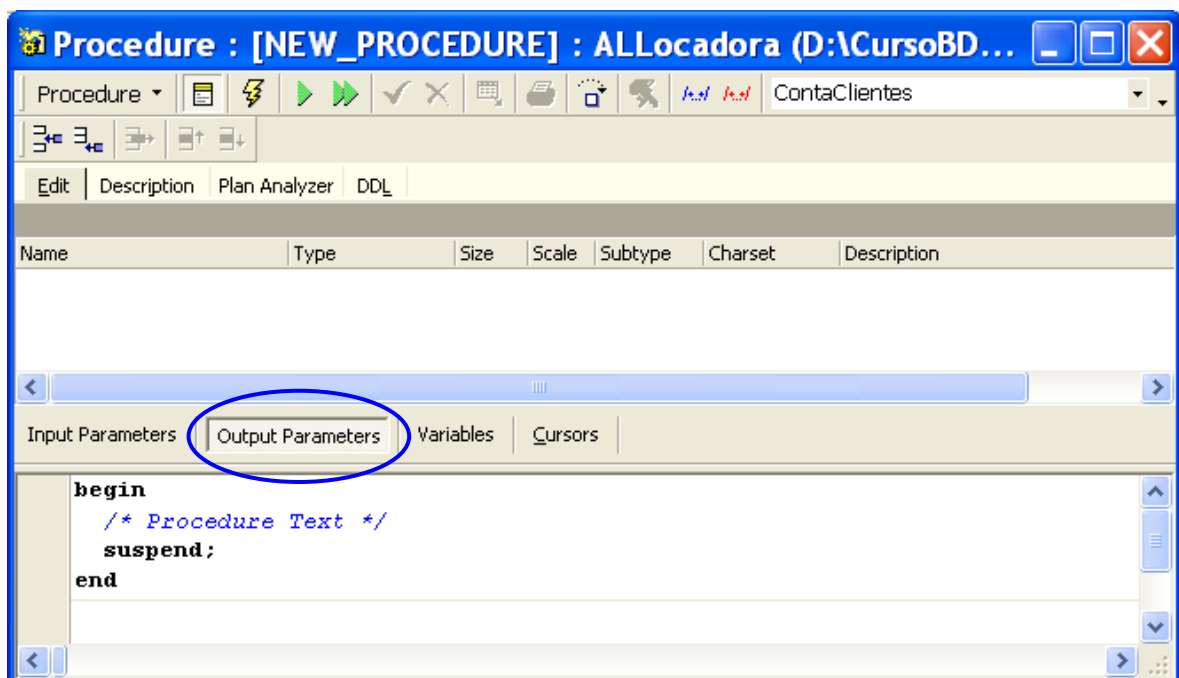



Figura 5.61a - Janela de criação de uma nova *stored procedure*

2º) Clique na opção “**Output Parameters**” para definir quais serão os parâmetros de saída; no nosso caso apenas um: a quantidade de clientes na tabela “Clientes”. É claro que, se existissem parâmetros de entrada, deveríamos defini-los na opção “Input Parameters”. Em seguida crie esse parâmetro; no caso “QuantCli” clicando no ícone de inserção do parâmetro .

Atenção: Se não clicar nesse ícone de inserção, o parâmetro não será criado. Veja **Figura 5.61b**.

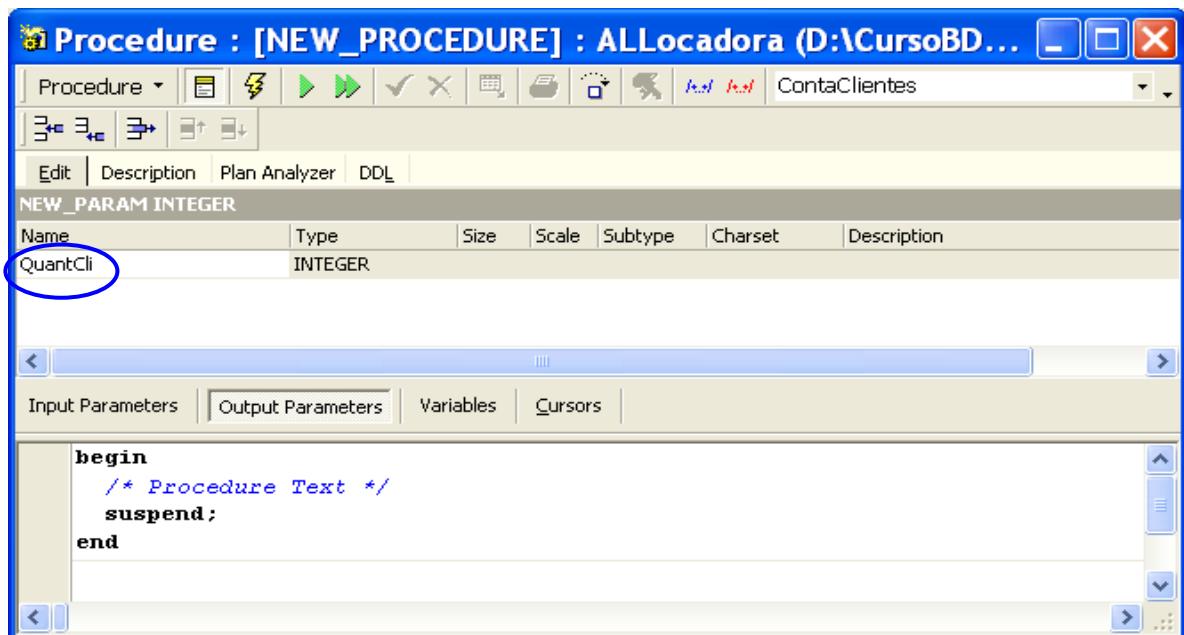


Figura 5.61b - Janela de criação de um novo parâmetro de saída

3º) Depois de criado os parâmetros (apenas um de saída no nosso exemplo que representa o retorno) da *stored procedure*, basta digitar o corpo principal em */* Procedure Text */* indicado na **Figura 5.61b**. O resultado é mostrado na **Figura 5.61c**.

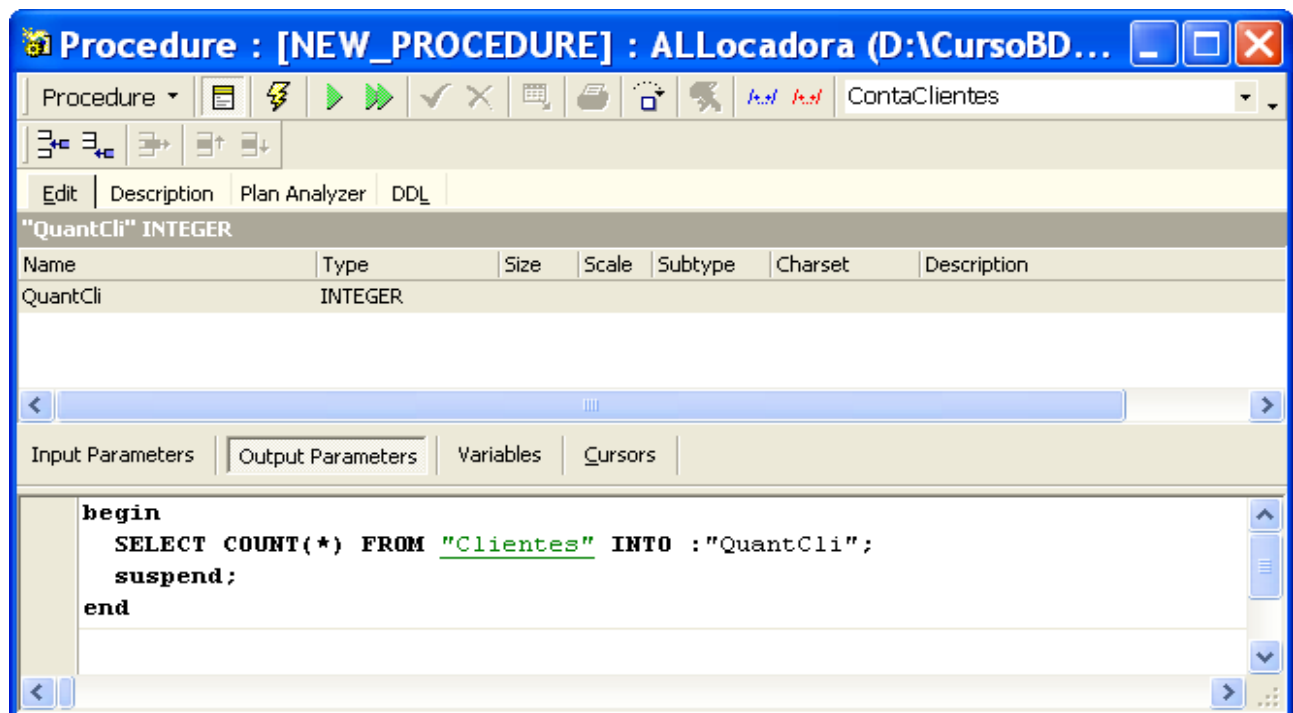



Figura 5.61c - O corpo da *stored procedure* digitada

4º) Definida a *stored procedure* ela deve ser compilada: clique em **Procedure/Compie procedure** ou clique diretamente no ícone , ou ainda alternativamente, no atalho **Ctrl+F9**; se não houver nenhum erro de sintaxe nas instruções sql a janela da **Figura 5.61d** aparece para que seja confirmada a compilação.

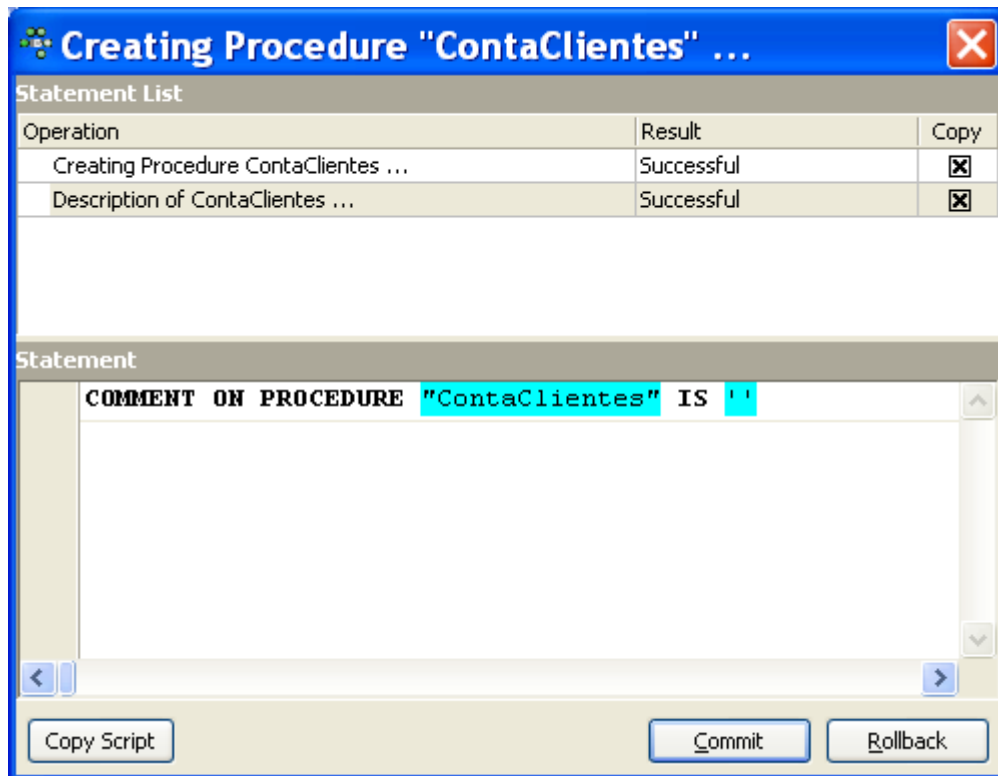


Figura 5.61d - A *stored procedure* pronta para ser criada

5º) Clique no botão **[Commit]** para confirmar a compilação. Pronto, assim a *stored procedure* estará criada e será exibida no “DB Explorer”; confira na **Figura 5.62**.

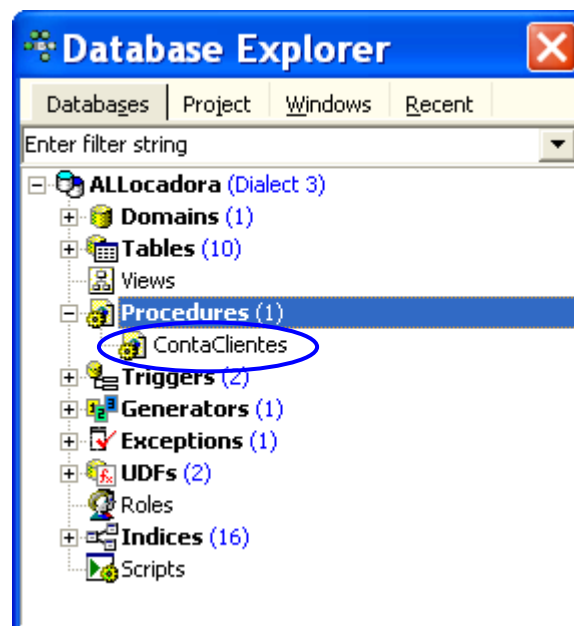



Figura 5.62 - A *stored procedure* exibida no “DB Explorer”

Executando a *stored procedure* “ContaClientes”

Agora que a *stored procedure* foi criada, podemos executá-la para conferir se está funcionando corretamente. Para isto basta abri-la (dando um duplo clique nela na janela do “DB Explorer”) e clicar no ícone de execução  na janela mostrada na **Figura 5.63a**.

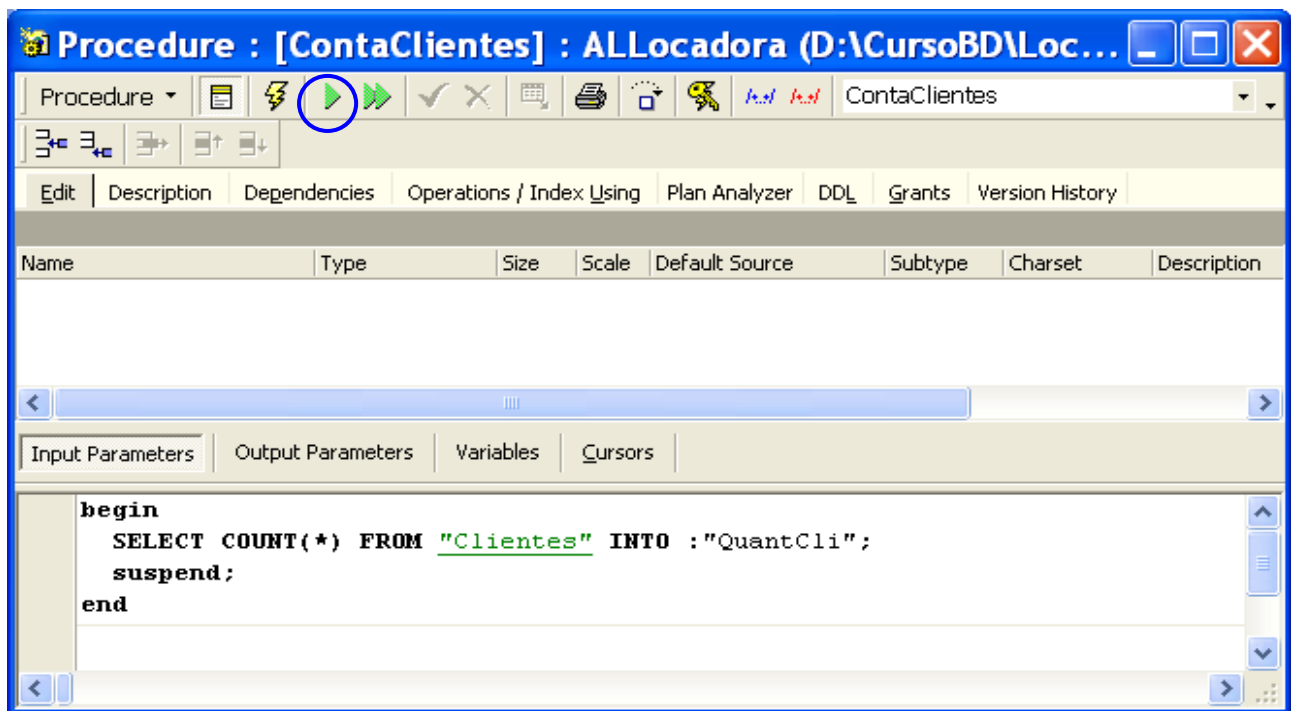


Figura 5.63a - A *stored procedure* pronta para ser executada...

Note o resultado na janela da **Figura 5.63b**, informando que atualmente existem 10 clientes...

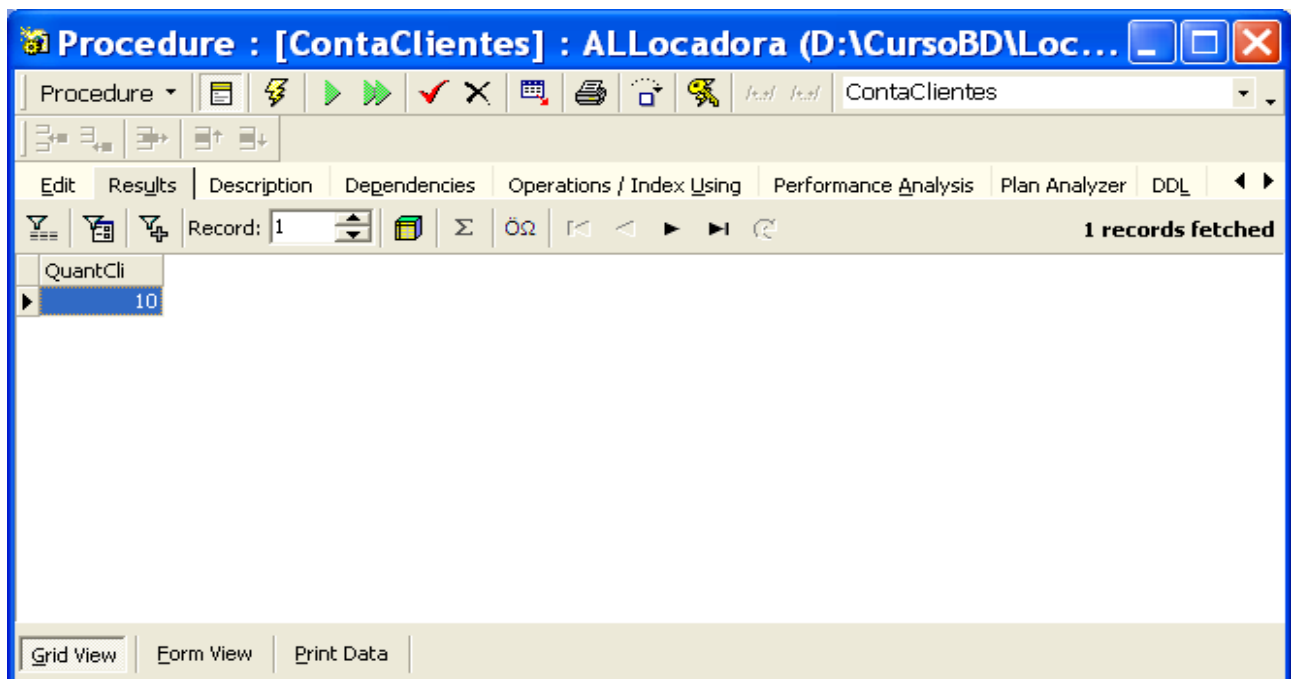


Figura 5.63b - A *stored procedure* dando o retorno (*resultado*) da sua execução

❖ Exemplo 5.5

Criar uma *stored procedure* denominada “ProcuraDir” que liste o código e o nome de todos os diretores de um determinado país.

Neste caso teremos que trabalhar com um parâmetro de entrada (o nome do país) e dois parâmetros de saída (código e nome do diretor). Também é importante frisar que nesta situação, devemos usar uma instrução contendo um *loop For...Do* para varrer toda a tabela “Diretor”; caso contrário o sistema gerará um erro dizendo que “existem muitas linhas”. Isto quer dizer que para retornar mais de uma linha na consulta à uma tabela, devemos sempre empregar um *loop*.

```
/*Selecionar todos os diretores de uma determinada nacionalidade, retornando
várias linhas... */
begin
  FOR SELECT "Dir_codigo", "Dir_nome" FROM "Diretor"
    WHERE "Dir_nacion" = :PNacion INTO :SDirCodigo, :SDirNome DO
    suspend;
end
```

Figura 5.64a mostra a definição do parâmetro de entrada; Figura 5.64b os parâmetros de saída.

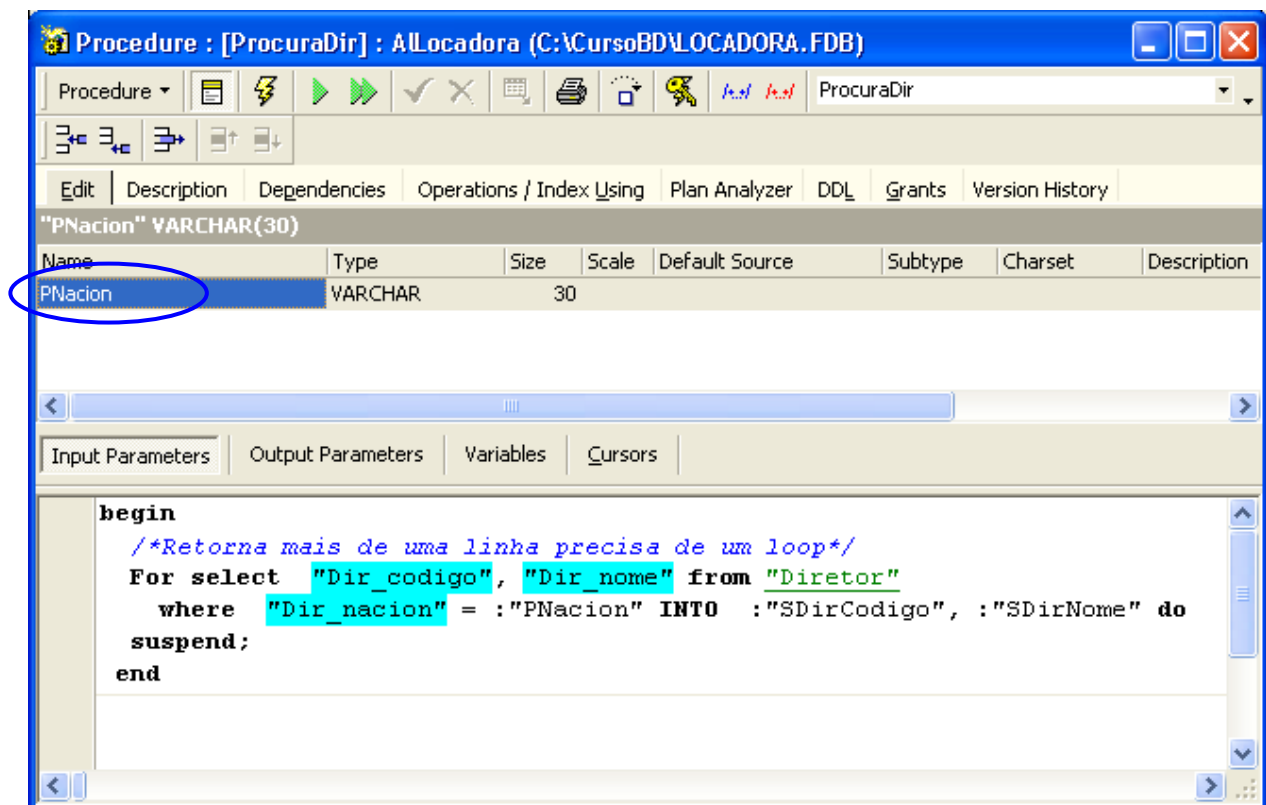


Figura 5.64a - A *stored procedure* destacando o parâmetro de entrada

ATENÇÃO: Dependendo da maneira como foi criado o banco de dados no IBEExpert, pode ser que seja necessário explicitar Tabela.campo (SEM ASPAS) na instrução sql. Por exemplo, no caso Exemplo 5.5, a instrução sql dentro da *stored procedure* “ProcuraDir” poderia ser assim:

```
begin
  FOR SELECT Diretor.dir_codigo, Diretor.Dir_nome FROM Diretor
    WHERE Diretor.dir_nacion = :PNacion INTO :SDirCodigo, :SDirNome DO
    suspend;
end
```

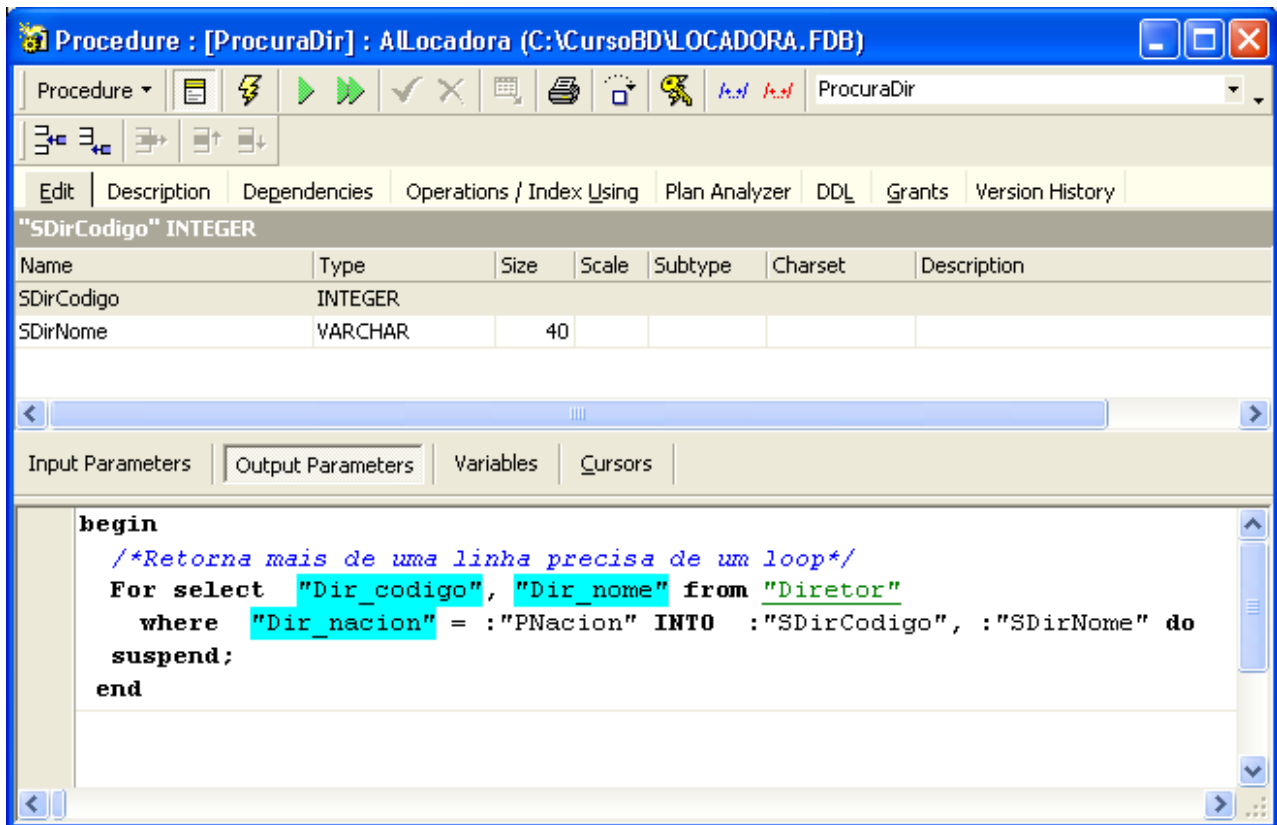



Figura 5.64b - A *stored procedure* destacando os parâmetro de saída

O parâmetro de entrada ("PNacion") receberá o nome do país a ser pesquisado na tabela "Diretor", e os parâmetros de saída ("SDirCodigo" e "SDirNome") exibirão os códigos e os nomes dos diretores desse país pesquisado, respectivamente. Clicando no ícone  para compilar a instrução, e em seguida *commitando* no botão **[Commit]** da janela que aparece, o resultado é uma janela a partir da qual a instrução poderá ser executada para se obter o resultado. Observe a Figura 5.65a.

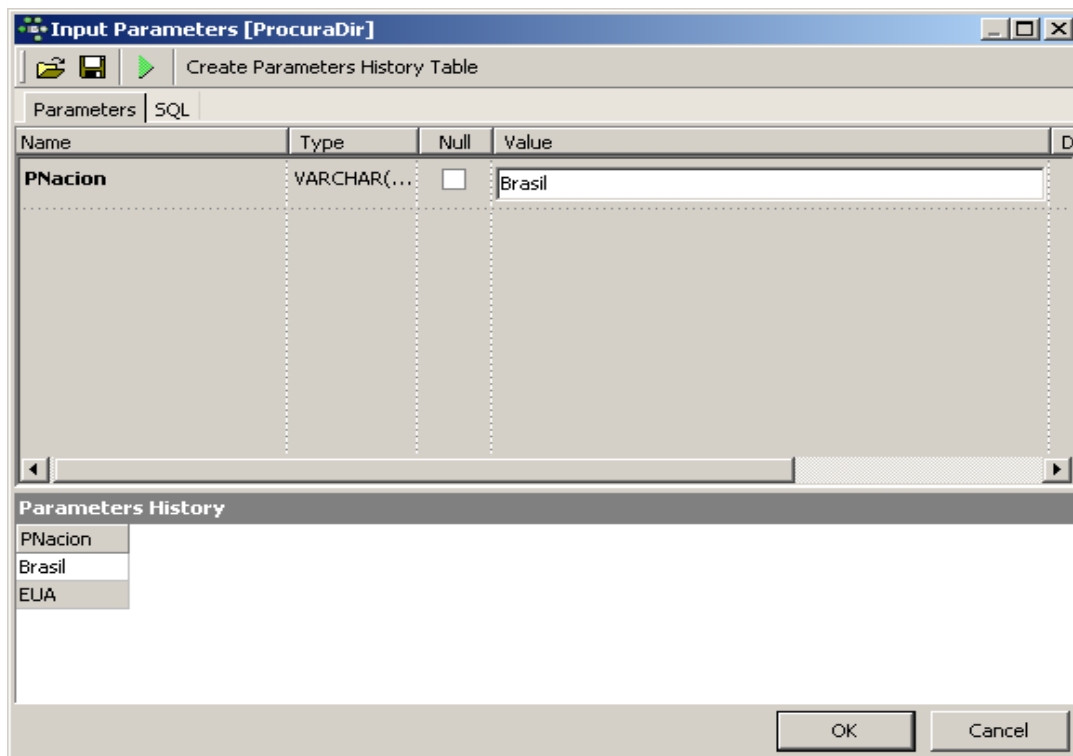


Figura 5.65a - Janela de execução da *stored procedure* "ProcuraDir"

Observe na **Figura 5.65a** que o parâmetro de entrada deve ser digitado na parte debaixo da janela, em “*Parameters History*”. Escolhendo “Brasil” obteremos todos os diretores do país; veja o resultado na **Figura 5.65b**.

SDirCodigo	SDirNome
10	Rogério Sganzerla
32	Carlos Manga
37	Carlos Diegues
40	Anselmo Duarte
43	Daniel Filho
49	Arnaldo Jabour
53	Nelson Pereira dos Santos
57	Júlio Bressane
59	Glauber Rocha
60	Bruno Barreto
62	Joaquim Pedro de Andrade
64	Domingos de Oliveira
69	Carlos Reichenbach
71	Tizuka Yamasaki
73	Walter Hugo Khouri

Figura 5.65b - Resultado da execução da *stored procedure* “ProcuraDir”

5.14 - Execução de instruções sql com o IBExpert

5.14.1 - Inserção de registros em tabelas

Utilizando o IBExpert (ou fazendo manualmente) podemos executar instruções sql num banco Firebird. Por exemplo, podemos incluir registros em tabelas, pesquisar registros, alterar registros, atualizar campos, etc. Entretanto, é importante ter muito cuidado, pois existem instruções com comandos não são aceitos pelo Firebird; mas, de um modo geral a maioria das instruções é compilada por esse servidor.

Como exemplo, vamos preencher as tabelas “Diretor”, “Filmes”, “Gênero”, “TipoFilme”, “Atores” e “AtorFilme” com instruções sql através do comando INSERT INTO.

A janela da **Figura 5.66** mostra o arquivo “InserDiretor.txt” criado com o “Bloco de Notas” do Windows XP; o seu conteúdo é composto de instruções sql para inserir dados dos diretores de filmes; embora a figura mostre dez diretores, o arquivo contém bem mais...

```

InsereDiretor - Bloco de notas
Arquivo  Editar  Formatar  Exibir  Ajuda

INSERT INTO "Diretor"
("dir_codigo", "dir_nome", "dir_nacion")
values (1, 'Steven Spielberg', 'EUA');

INSERT INTO "Diretor"
("dir_codigo", "dir_nome", "dir_nacion")
values (2, 'Frederico Fellini', 'Itália');

INSERT INTO "Diretor"
("dir_codigo", "dir_nome", "dir_nacion")
values (3, 'Alfred Hitchcock', 'Inglaterra');

INSERT INTO "Diretor"
("dir_codigo", "dir_nome", "dir_nacion")
values (4, 'George Lucas', 'EUA');

INSERT INTO "Diretor"
("dir_codigo", "dir_nome", "dir_nacion")
values (5, 'Mel Brooks', 'EUA');

INSERT INTO "Diretor"
("dir_codigo", "dir_nome", "dir_nacion")
values (6, 'Francis Ford Coppola', 'EUA');

INSERT INTO "Diretor"
("dir_codigo", "dir_nome", "dir_nacion")
values (7, 'Martin Scorsese', 'EUA');

INSERT INTO "Diretor"
("dir_codigo", "dir_nome", "dir_nacion")
values (8, 'Ingmar Bergman', 'Suécia');

INSERT INTO "Diretor"
("dir_codigo", "dir_nome", "dir_nacion")
values (9, 'Pedro Almodóvar', 'Espanha');

INSERT INTO "Diretor"
("dir_codigo", "dir_nome", "dir_nacion")
values (10, 'Rogério Sganzerla', 'Brasil');

Ln 1085, Col 1

```

Figura 5.66 - Arquivo “InsereDiretor.txt” com as instruções de inserir registros na tabela “Diretor”

ATENÇÃO: Note na **Figura 5.66** que o nome da tabela e os nomes dos campos foram colocados entre aspas (“ ”) e os conteúdos do tipo *string* entre apóstrofos (‘ ’). Isto deve ser observado, pois senão o executor de instruções sql do IBExpert não aceita para esse banco como foi criado. Também, aqui NÃO vamos usar o “SQL Editor” do IBExpert, pois ele só está preparado para executar uma (apenas uma) instrução de cada vez; assim, se tentarmos inserir vários registros na tabela “Diretor” com vários comandos INTO INSERT, observe na **Figura 5.67** a janela com a mensagem de erro que ocorre quando tentamos fazer isto! A solução é utilizar um outro gerenciador de instruções sql/ oferecido pelo IBExpert: “**Script Executive**”.

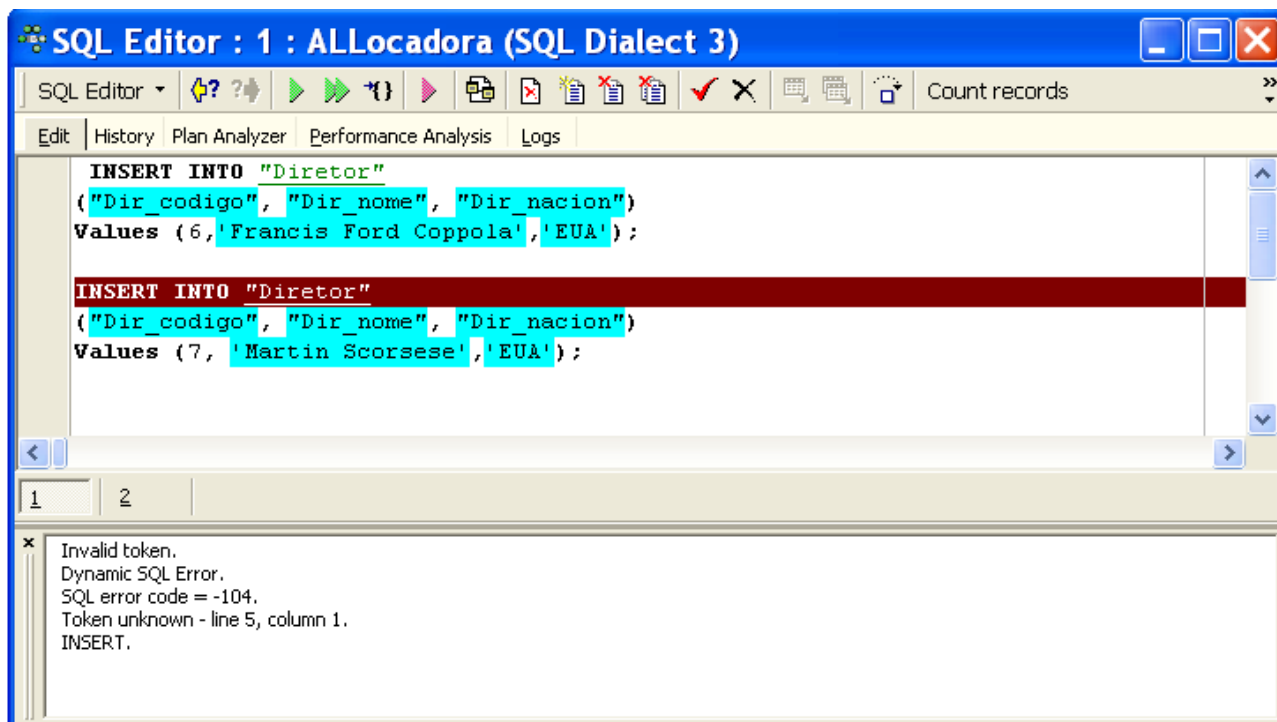


Figura 5.67 - Erro ao tentar inserir mais de um registros usando o “SQL Editor” do IBERP

Portanto, quando houver necessidade de executar mais de uma instrução *sql* no ambiente do IBERP, deve ser utilizado o “**Script Executive**” e não o “SQL Editor”. Para isto execute os seguintes passos:

1º) Clique em **Tools** no *menu* principal e em seguida selecione a opção “**Script Executive**”, ou via teclas de atalho com **Ctrl+F12**, conforme indica a Figura 5.68a.

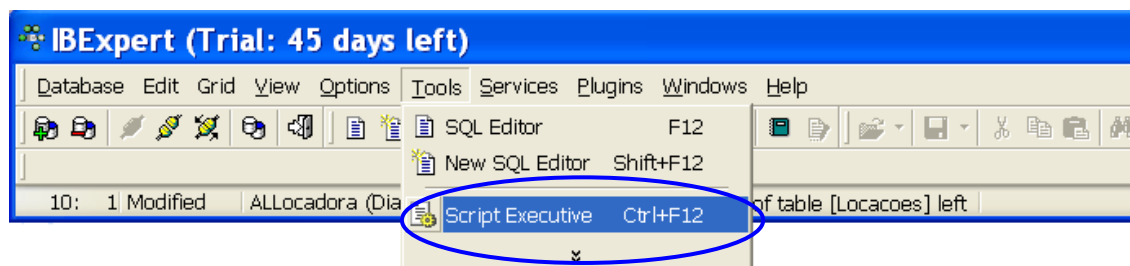


Figura 5.68a - Carregando o “Script Executive”

2º) Copie as instruções *sql* do arquivo “InserDiretor.txt” e cole-as dentro do “Script Executive”. Em seguida, ANTES DE EXECUTAR AS INSTRUÇÕES, clique em “**Script**” e execute a opção “**Add CONNECT statement**” para *conectar* o banco de dados. Observe na Figura 5.68b.

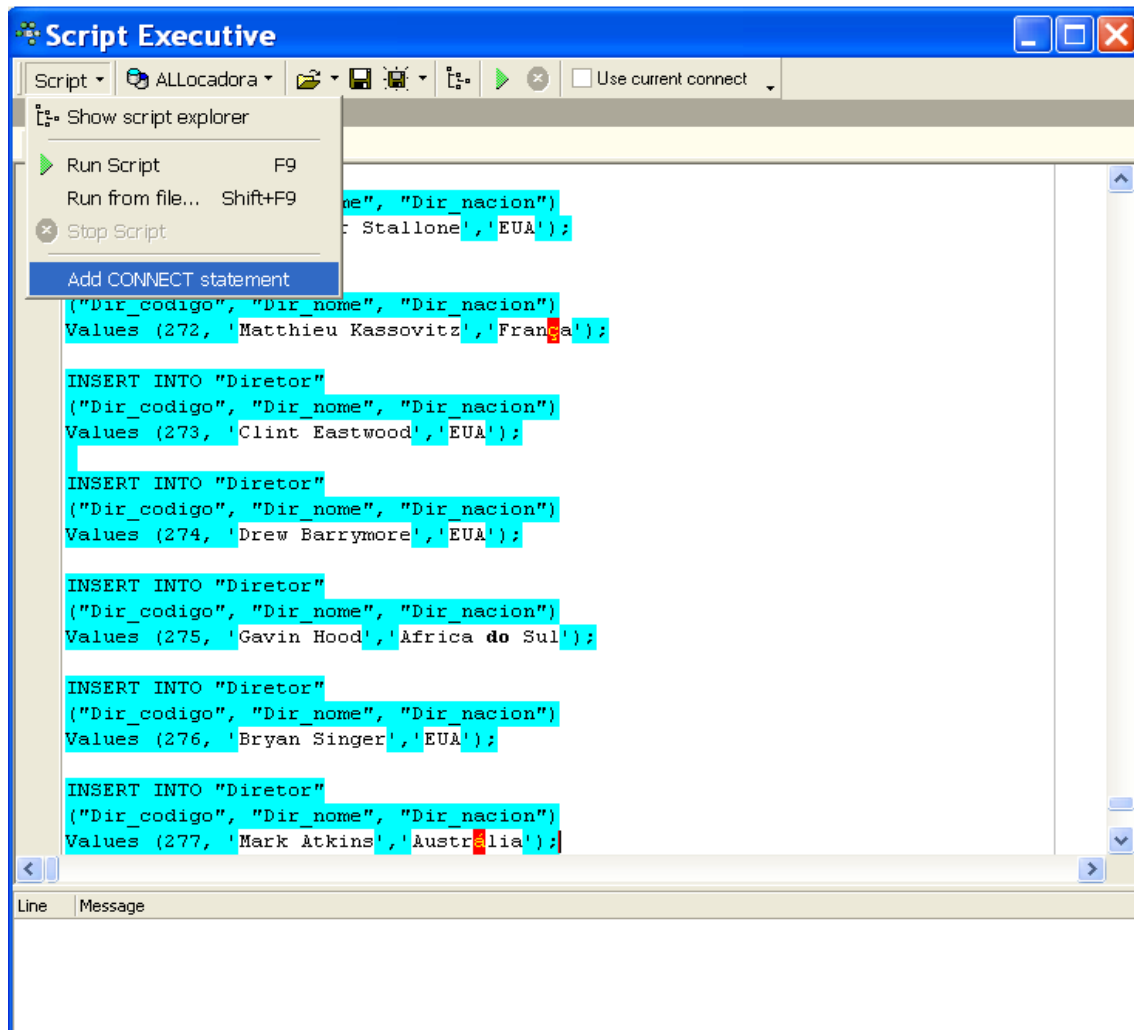


Figura 5.68b - Conectando o banco de dados ANTES de executar as instruções *sql*

3º) Ao conectar o banco de dados uma janela, como a da **Figura 5.68c**, será exibida para que seja confirmada a conexão utilizando o dialeto sql 3 definido por ocasião da criação do banco de dados.

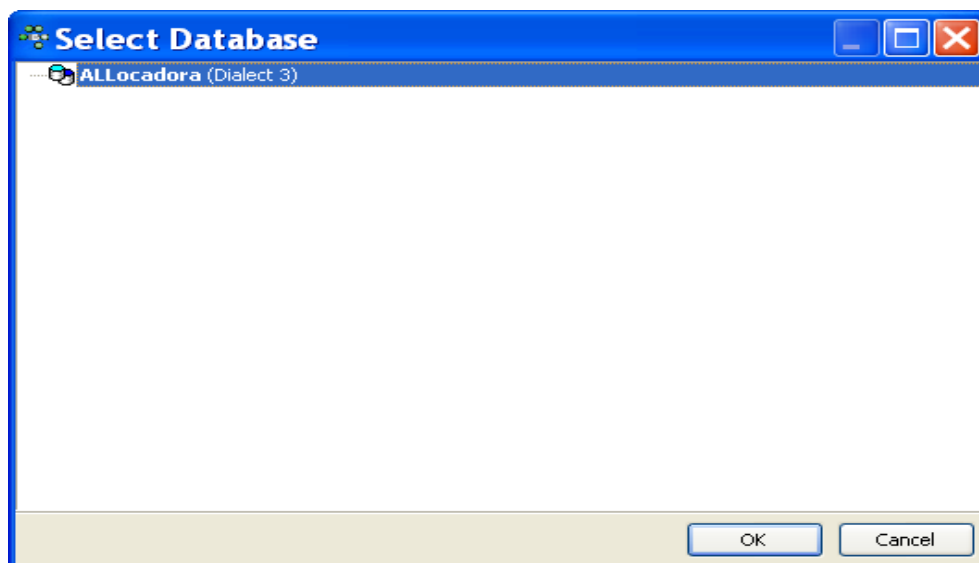


Figura 5.68c - Janela de confirmação da conexão com o banco de dados

4º) Confirmando a conexão com o banco de dados, a janela da **Figura 5.68d** será apresentada, mostrando na primeira linha o caminho de localização do banco, e em seguida as linhas de instruções para inserir os registros.

Obs) Caso não tivesse feito a conexão, ao tentar executar as instruções seria gerada a seguinte mensagem de erro: *“Cannot perform operation ... DB is not open”*. Atente bem para esse detalhe!

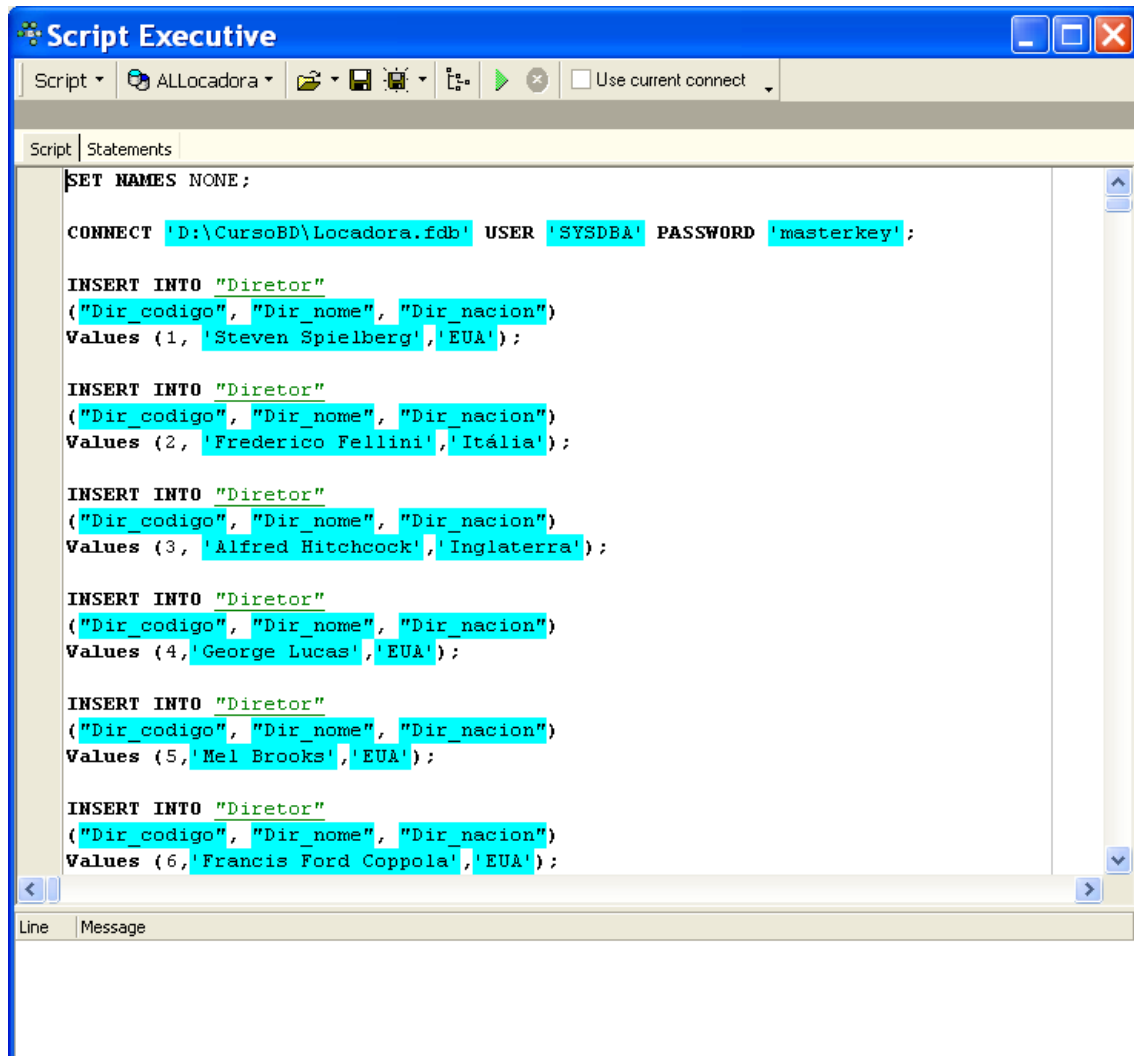



Figura 5.68d - Janela do “Script Executive” com as instruções *sql*

5º) Finalmente, podemos executar as instruções para que sejam inseridos na tabela “Diretor” todos os registros desejados. Para isto basta clicar em **“Script/Run Script”** ou diretamente no ícone , ou ainda utilizando as teclas de atalho **Ctrl+F9**. Observe o resultado na **Figura 5.69**, com a tabela “Diretor” exibindo os registros inseridos.

Dir_codi...	Dir_nome	Dir_nacion
1	Steven Spielberg	EUA
2	Frederico Fellini	Itália
3	Alfred Hitchcock	Inglaterra
4	George Lucas	EUA
5	Mel Brooks	EUA
6	Francis Ford Coppola	EUA
7	Martin Scorsese	EUA
8	Ingmar Bergman	Suécia
9	Pedro Almodóvar	Espanha
10	Rogério Sganzerla	Brasil
11	Bernardo Bertolucci	Itália
12	Oliver Stone	EUA
13	Alice Guy-Blaché	França
14	Woody Allen	EUA
15	Spike Lee	EUA
16	John Ford	EUA
17	David Cronenberg	Canadá
18	Jean-Luc Godard	França
19	Akira Kurosawa	Japão
20	Quentin Tarantino	Espanha
21	Roberto Rossellini	Itália
22	Sydney Pollack	EUA
23	Jean-Pierre Jeunet	França
24	John Boorman	França

Figura 5.69 - Janela da tabela “Diretor” com seus registros

Agora, as outras tabelas também poderão ser preenchidas do mesmo modo como foi feito para a tabela “Diretor”.

5.14.2 - Criação de tabelas

A instrução *sql* para criar uma tabela tem a seguinte sintaxe geral:

```
CREATE TABLE Nome_Tabela
(Campo1 Tipo [complemento],
 Campo2 Tipo [complemento],
 Campo3 Tipo [complemento],
 . . .
 CampoN Tipo [complemento],
PRIMARY KEY (campo),
CONSTRAINT FOREIGN KEY Nome_FK1 (campo) REFERENCES Tabela-mãe1 (PK),
CONSTRAINT FOREIGN KEY Nome_FK2 (campo) REFERENCES Tabela-mãe2 (PK),
CONSTRAINT FOREIGN KEY Nome_FK2 (campo) REFERENCES Tabela-mãe3PK),
. . .
);
```

Onde [complemento] são as possíveis cláusulas de configurações para o campo, tais como:

SET DEFAULT: indica valor padrão para o campo

NOT NUL: indica que um valor para o campo é obrigatório.

UNIQUE: indica que o campo deve ter valor único, não repetido.

AUTO INCREMENT: se o campo é numérico se ele vai ser auto incrementado.

A palavra-chave **CONSTRAINT** permite criar as restrições de chaves estrangeiras.

Para diversificar nossos exemplos, vamos criar um novo banco de dados no servidor Firebird; esse novo banco será denominado: “Escola.fdb”, baseado no MER da **Figura 5.70**.

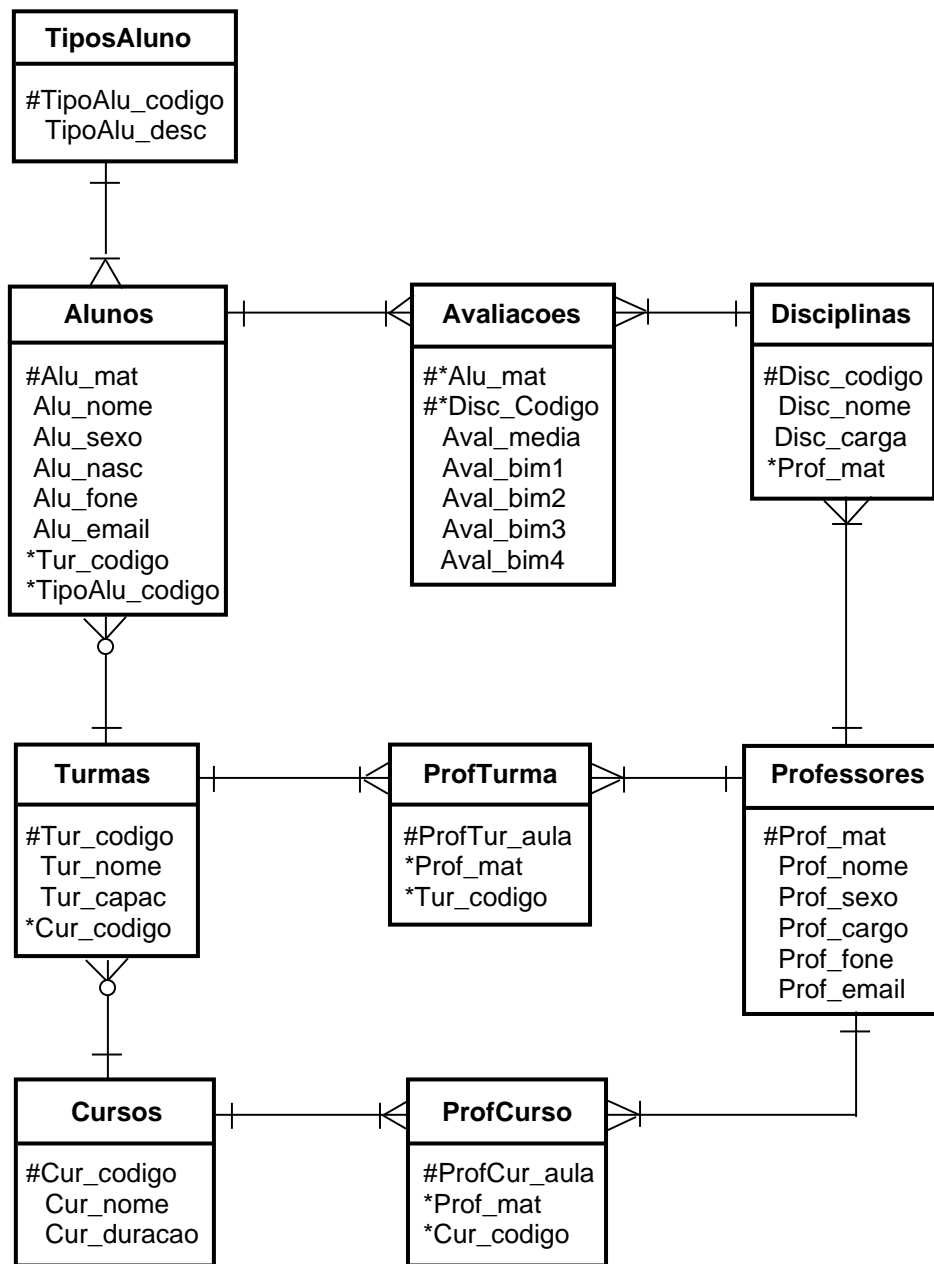


Figura 5.70 - MER do banco de dados “Escola.fdb”

A tabelas “Avaliacoes”, “ProfTurma”, e “ProfCurso” são tabelas associativas dos relacionamentos entre “Alunos-Disciplinas”, “Professores-Turmas” e Professores-Cursos”, respectivamente. Estamos supondo também que uma determinada disciplina só pode ser dada por um único professo (o que nos parece óbvio, já que, por exemplo, “Matemática I” é considerada uma disciplina distinta de “Matemática II, mesmo que ambas tratem do mesmo assunto: Matemática)

Então, usando apenas instruções *sql* (executadas dentro do “SQL Editor”) vamos criar as tabelas do banco “Escola.fdb”, baseado no MER da **Figura 5.70**. A **Figura 5.71a** mostra o novo banco de dados criado no servidor (ainda não conectado).



Figura 5.71a - O novo banco exibido no “DB Explorer”

O banco “Escola.fdb” (representado pelo seu *alias* ALEscola”) ainda não possui nenhum elemento. Vamos criar as tabelas e chaves com as instruções a seguir, obedecendo o critério de que as tabelas-filhas só podem ser criadas depois das respectivas tabelas-mãe.

Primeiramente devemos dar *login* no banco de dados “ALEscola”, pois a **Figura 5.71a** mostra que apesar de estar criado no servidor, ele ainda não está *logado*. Dê um duplo clique sobre ele e entre com o *login* (SYSDBA) e a *password* (masterkey). E por garantia, vamos desconectar o banco “ALocadora” clicando com o botão direito sobre ele e em seguida executando a opção **“Disconnect from Database”**, o resultado é mostrado na **Figura 5.71b**, indicando que apenas o banco “ALEscola” está conectado agora.

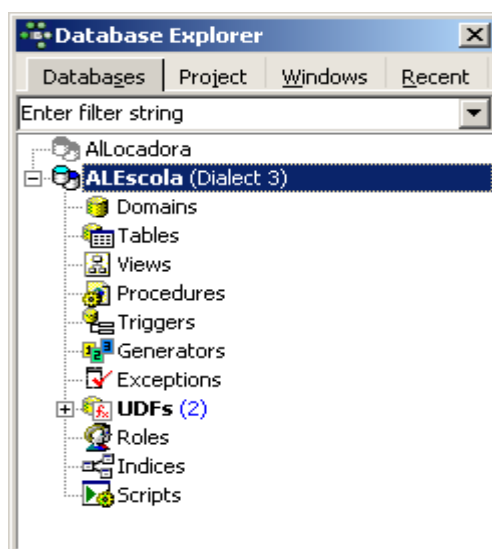


Figura 5.71b - O novo banco *logado* no servidor

Agora sim, podemos criar as tabelas com segurança...

1) Tabela "Professores"

```
CREATE TABLE Professores
(
  Prof_mat      INTEGER NOT NULL ,
  Prof_nome     VARCHAR(40) NOT NULL,
  Prof_sexo     CHAR(1) ,
  Prof_cargo    VARCHAR(50) ,
  Prof_email    CHAR(50) ,
  Prof_fone     CHAR(13) ,
  PRIMARY KEY(Prof_mat)
);
```

2) Tabela "Cursos"

```
CREATE TABLE Cursos
(
  Cur_Codigo    VARCHAR(6) NOT NULL,
  Cur_nome      VARCHAR(40) ,
  Cur_duracao   SMALLINT,
  PRIMARY KEY(Cur_Codigo)
);
```

3) Tabela "TiposAluno"

```
CREATE TABLE TiposAluno
(
  TipoAlu_Codigo SMALLINT NOT NULL,
  TipoAlu_Desc   VARCHAR(20) ,
  PRIMARY KEY(TipoAlu_Codigo)
);
```

4) Tabela "Turmas"

```
CREATE TABLE Turmas
(
  Tur_codigo    VARCHAR(8) NOT NULL,
  Tur_nome      VARCHAR(50) ,
  Tur_capac     SMALLINT,
  Cur_codigo    VARCHAR(6) NOT NULL,
  PRIMARY KEY(Tur_codigo),
  CONSTRAINT FK_Curso1 FOREIGN KEY(Cur_codigo) REFERENCES Cursos(Cur_codigo)
);
```

5) Tabela “Disciplinas”

```

CREATE TABLE Disciplinas
(
  Disc_codigo  VARCHAR(6) NOT NULL,
  Disc_nome    VARCHAR(40) NOT NULL,
  Disc_carga   SMALLINT,
  Prof_mat     INTEGER NOT NULL,
  PRIMARY KEY(Disc_codigo),
  CONSTRAINT FK_Prof1 FOREIGN KEY(Prof_mat) REFERENCES Professores(Prof_mat)
);

```

6) Tabela “Alunos”

```

CREATE TABLE Alunos
(
  Alu_mat      INTEGER NOT NULL ,
  Alu_nome     VARCHAR(40) NOT NULL,
  Alu_sexo     CHAR(1) ,
  Alu_nasc     DATE,
  Alu_fone     VARCHAR(13) ,
  Alu_email    VARCHAR(50) ,
  Tur_codigo   VARCHAR(6) ,
  TipoAlu_codigo SMALLINT,
  PRIMARY KEY(Alu_mat) ,
  CONSTRAINT FK_Turmal FOREIGN KEY(Tur_codigo) REFERENCES Turmas(Tur_codigo) ,
  CONSTRAINT FK_TpAlu FOREIGN KEY(TipoAlu_codigo) REFERENCES TiposAluno(TipoAlu_codigo)
);

```

7) Tabela “Avaliacoes”

```

CREATE TABLE Avaliacoes
(
  Alu_mat      INTEGER NOT NULL ,
  Disc_codigo  VARCHAR(6) NOT NULL,
  Ava_media    DECIMAL(5,2) ,
  Ava_bim1     DECIMAL(5,2) ,
  Ava_bim2     DECIMAL(5,2) ,
  Ava_bim3     DECIMAL(5,2) ,
  Ava_bim4     DECIMAL(5,2) ,
  PRIMARY KEY(Alu_mat, Disc_codigo) ,
  CONSTRAINT FK_Aluno FOREIGN KEY(Alu_mat) REFERENCES Alunos(Alu_mat) ,
  CONSTRAINT FK_Disc FOREIGN KEY(Disc_codigo) REFERENCES Disciplinas(Disc_codigo)
);

```


8) Tabela “ProfCurso”

```
CREATE TABLE ProfCurso
(
  ProfCur_aula INTEGER NOT NULL,
  Prof_mat      INTEGER NOT NULL,
  Cur_codigo    VARCHAR(6) NOT NULL,
  PRIMARY KEY (ProfCur_aula),
  CONSTRAINT FK_Prof2 FOREIGN KEY (Prof_mat) REFERENCES Professores(Prof_mat),
  CONSTRAINT FK_Curso2 FOREIGN KEY (Cur_codigo) REFERENCES Cursos(Cur_codigo)
);
```

9) Tabela “ProfTurma”

```
CREATE TABLE ProfTurma
(
  ProfTur_aula INTEGER NOT NULL,
  Prof_mat      INTEGER NOT NULL,
  Tur_codigo    VARCHAR(8) NOT NULL,
  PRIMARY KEY (ProfTur_aula),
  CONSTRAINT FK_Prof3 FOREIGN KEY (Prof_mat) REFERENCES Professores(Prof_mat),
  CONSTRAINT FK_Turma2 FOREIGN KEY (Tur_codigo) REFERENCES Turmas(Tur_codigo)
);
```

Para criar essas tabelas, basta copiar os códigos, colá-los dentro do “SQL Editor” (um de cada vez) e executar. Por exemplo, a **Figura 5.72** mostra a instrução de criação da tabela “Professores” dentro do editor de instruções *sql*.

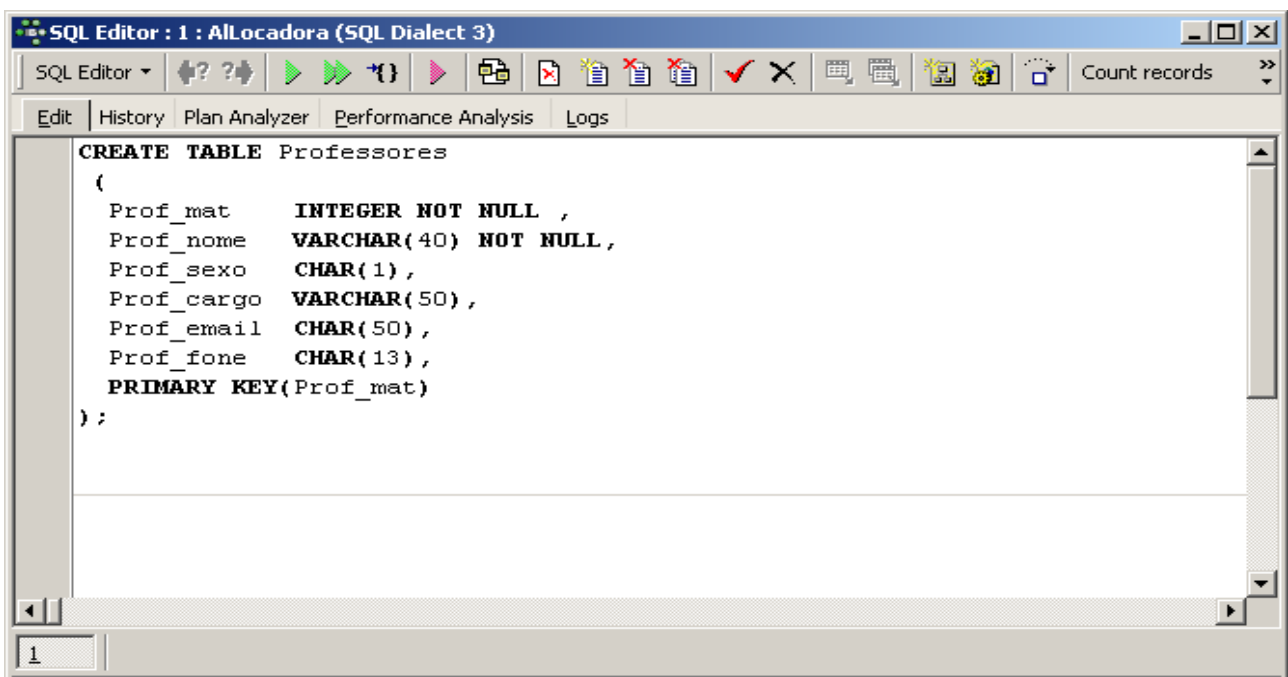




Figura 5.72 - Script da instrução de criação da tabela “Professores”

Nota: Palavras-chaves são apresentadas em **negrito** dentro do editor de instruções *sql*. Assim, se ao digitar uma dessas palavras e não ficar **negritada** é porque ela foi não foi ou não está sendo digitada corretamente. Isto é uma boa técnica para validar textos dentro de um editor, e nesse caso é muito importante utilizar esse recurso.

Para executar a instrução basta clicar no ícone  e em seguida *commitar* clicando em . Pronto; se não houver nenhum erro de sintaxe na instrução, a tabela estará criada. Isto deve ser feito para todas as outras oito tabelas. A **Figura 5.73** mostra nosso banco de dados “Escola.fdb” com todas as nove tabelas indicadas pelo MER da **Figura 5.70**.

Ao executar essas instruções (considerando a prioridade para as tabelas-mães nos relacionamentos), o “DB Explorer” exibirá esses elementos tal como mostra a **Figura 5.73**.

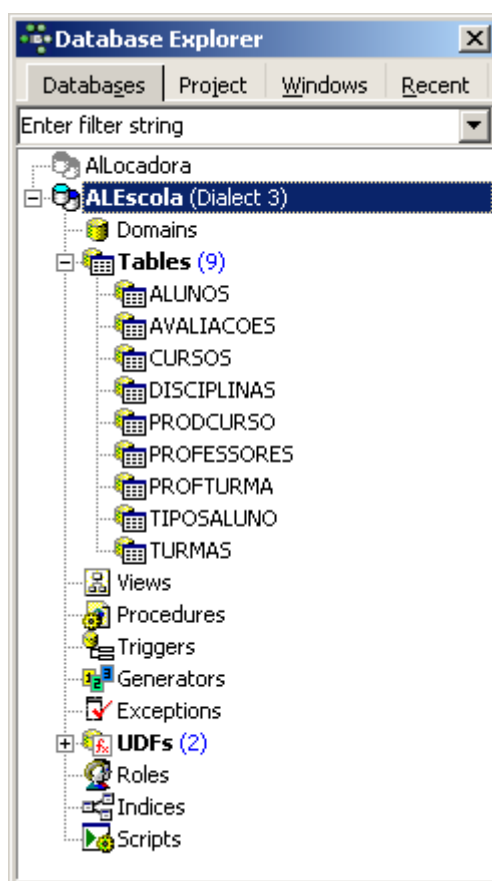


Figura 5.73 - O banco “Escola.fdb” com suas tabelas

Nota: Para este novo banco usamos a configuração do **Charset** como **Win1252** (padrão para o Brasil e países do Leste Europeu). O que isto significa de tão importante?! Uma das consequências já foi sentida anteriormente: lembra da **Nota** logo após a **Figura 5.66**? Nela foi dito que os nomes dos campos deveriam ser escritos entre aspas (“ ”). Isto foi preciso na ocasião devido ao fato do banco “Locadora.fdb” não ter ido criado com o padrão Win1252. Se tivéssemos usados esse padrão, não haveria necessidade de colocar os nomes dos campos entre aspas. Os conteúdos dos campos *strings* sim, devem sempre ser colocados entre apóstrofes, não importa o padrão do *Charset* escolhido; mas os nomes dos campos não! Portanto, atente bem para esse detalhe na hora de criar um novo banco de dados. As **figura 5.74a e 5.74b** mostram como essa configuração deve ser feita; basta escolher o padrão **Win1252** na lista de *charsets* apresentada.

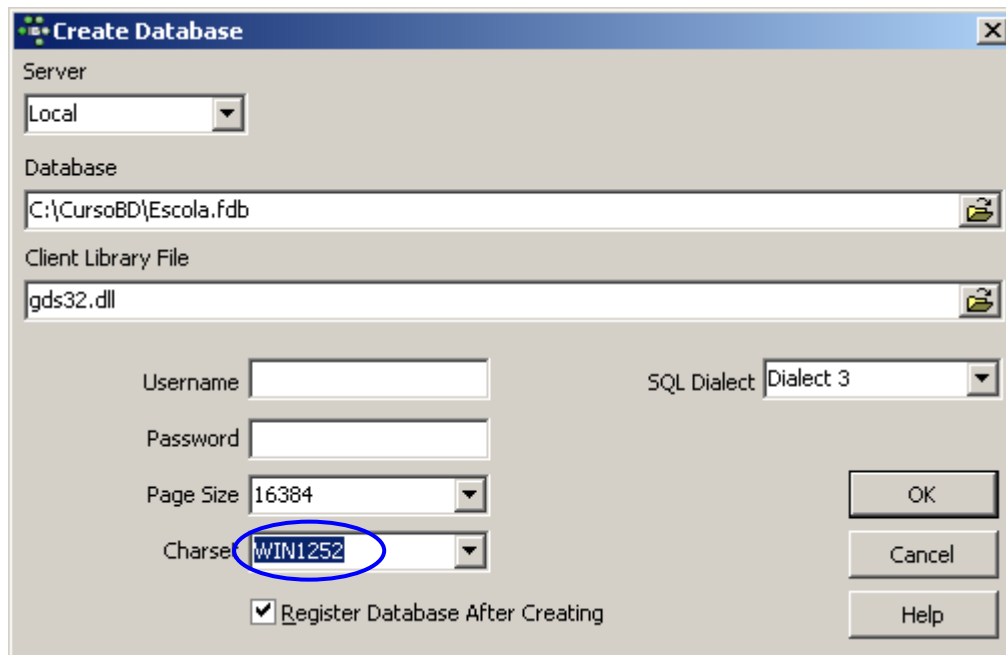


Figura 5.74a - Escolhendo o padrão *Charset* na criação do banco

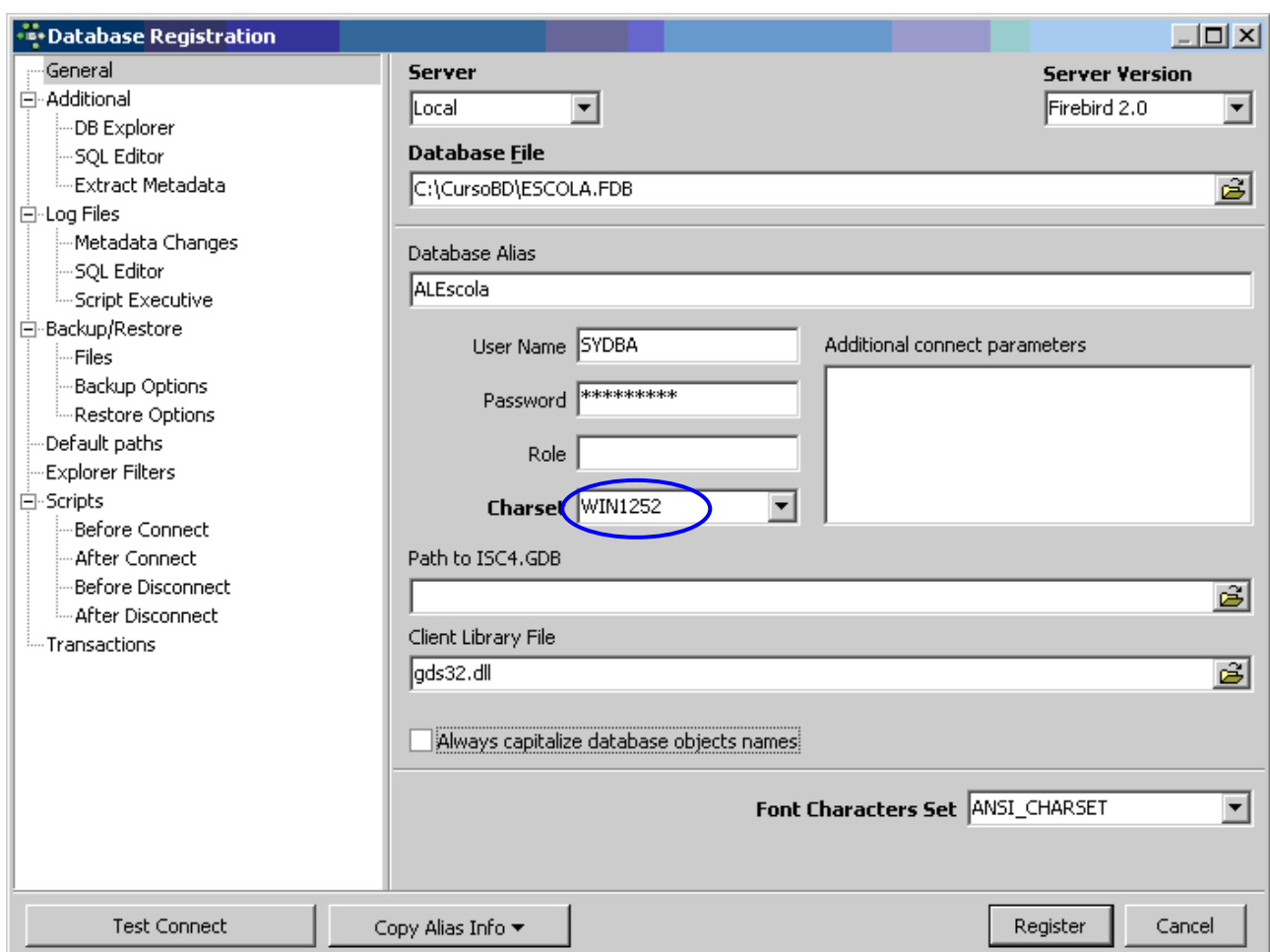


Figura 5.74b - Registrando o banco com o padrão *Charset* desejado

5.15 - Trabalhando com campos BLOB

Campos do tipo BLOB (**B**inary **L**arge **O**bject) raramente são citados e usados em bancos de dados, mas eles podem fazer grande diferença quando a tabela precisa armazenar textos muito longos e/ou imagens (fotos, figuras, etc). Nesses casos seu emprego é necessário. Depois de definido o campo como BLOB, seu uso é relativamente fácil. Para exemplificar, vamos supor que cada aluno do nosso banco de dados “Escola.fdb” deva ter sua foto armazenada na tabela “Alunos”; então é preciso criar um novo campo nessa tabela só para armazenar a foto.

1º) Criar um novo campo, por exemplo, “Alu_foto” para armazenar a foto do aluno. Abra a tabela “Alunos” e adicione um novo campo com esse nome. Observe que a guia “Raw Datatype” deve ser selecionada para definir os dados para esse novo campo. Veja a **Figura 5.75a**.

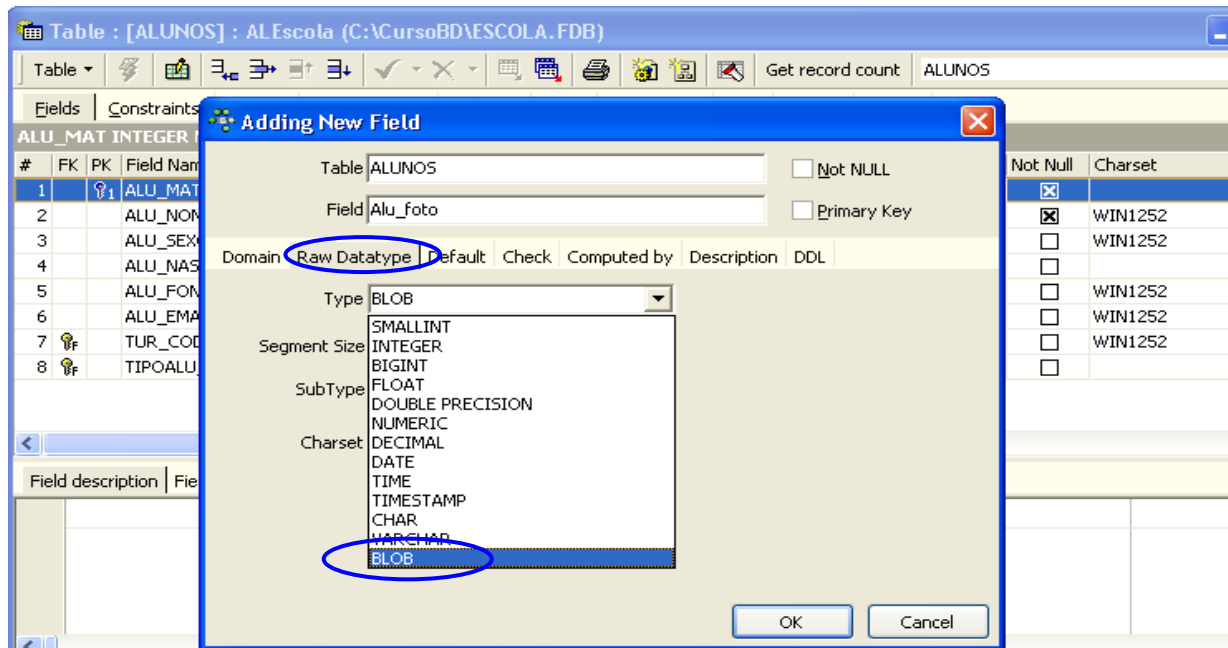


Figura 5.75a - Definindo as características do novo campo “Alu_foto”

2º) Confirme no botão [OK] e em seguida *commit* a operação; Assim o novo campo é inserido.

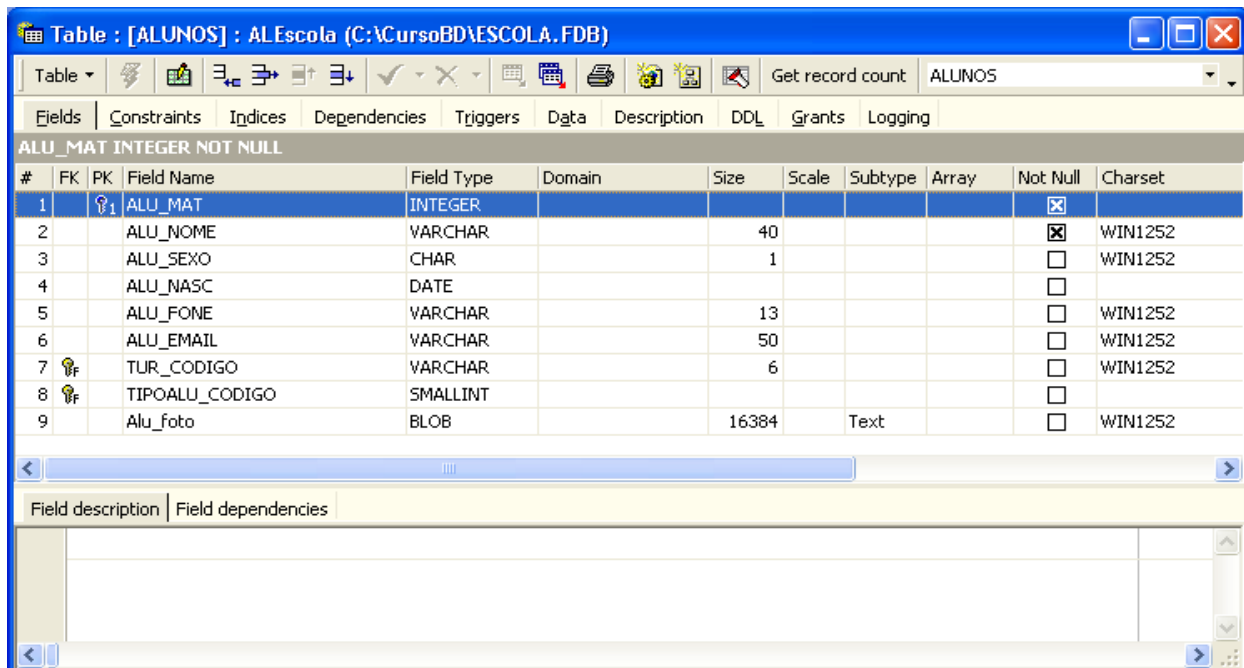


Figura 5.75b - A tabela “Alunos” agora exibindo o novo campo do tipo *BLOB*

Para entrar dados num campo BLOB clique na guia “Data” da **Figura 5.75b** e em seguida vá em no rodapé da janela e clique na guia “Form View”. Aparece uma janela como a da **Figura 5.76a**.

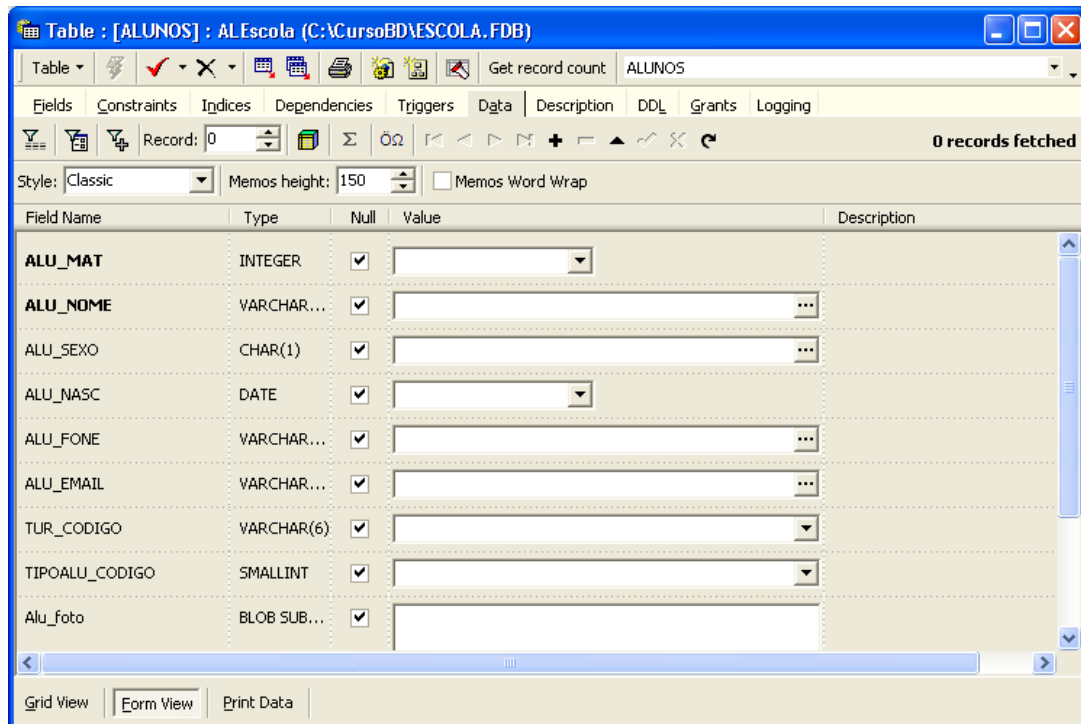


Figura 5.76a - A janela indicada com “Form View” pronta para receber dados

Observe que pela janela da **Figura 5.76a** é possível entrar com dados em todos os campos não só no campo BLOB. Entretanto, para entrar o campo BLOB é preciso fazer o seguinte:

1) Clique cm o botão direito do *mouse* sobre a parte branca do campo BLOB; aparecem as duas opções “Save do file” e “Load from file”. Clique sobre esta última para carregar a imagem.

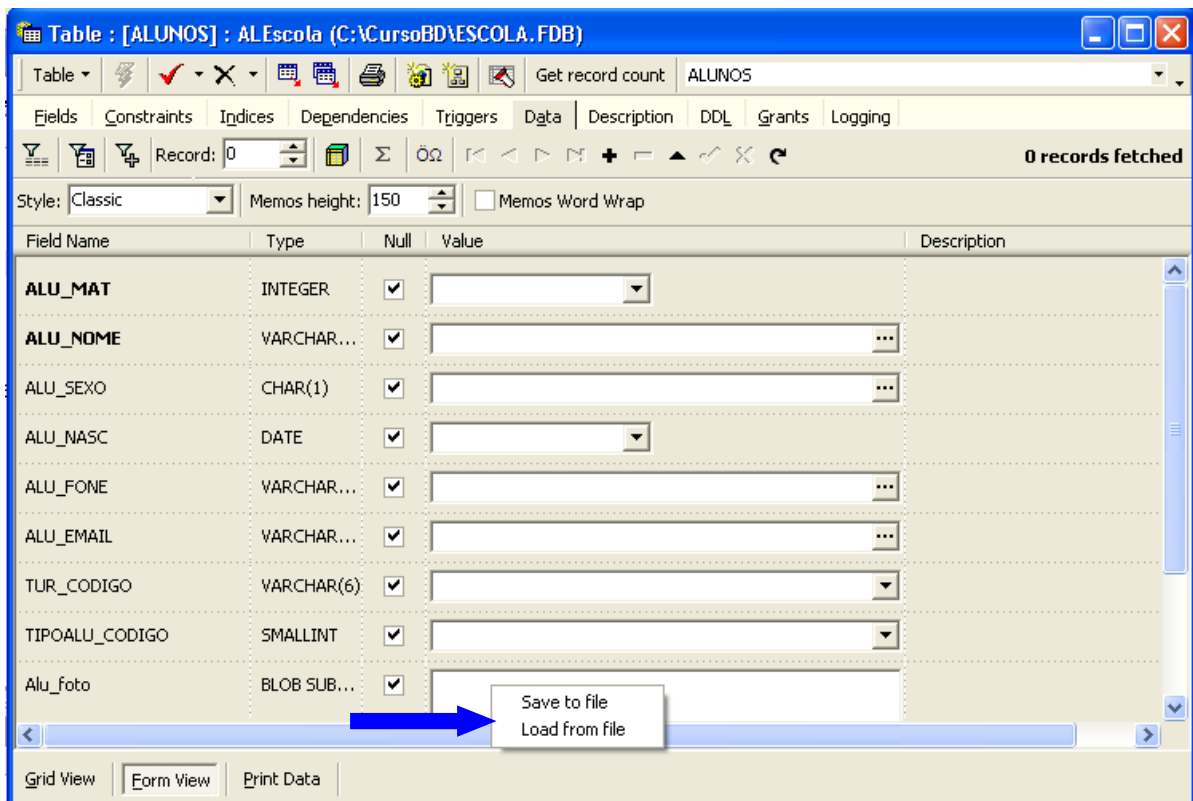


Figura 5.76b - Carregando a imagem com “Load from file”

3) Depois de selecionar a opção “Load from file” na tela anterior, carregue efetivamente o arquivo que contém a imagem na janelinha “Load Blod-Field Form File” que aparece, e pronto!

A **Figura 5.77a** mostra a inclusão de um aluno; observe que para os campos que são chaves estrangeiras aparece uma janelinha com os dados da tabela-mãe, facilitando a entrada do dado. No caso, para entrar com o código da turma, ao clicar na caixa *combo* aparece uma janeinha com os dados da tabela “Turma”. Então é só selecionar a turma desejada, evitando assim erros de digitação e/ou entradas incorretas. Para os campos numéricos aparece uma pequena calculadora para auxiliar na entrada de números.

Field Name	Type	Null	Value	Description
ALU_MAT	INTEGER		1	
ALU_NOME	VARCHAR...		Ana Paula Foz R. Leite	
ALU_SEXO	CHAR(1)		F	
ALU_NASC	DATE		07.08.1987	
ALU_FONE	VARCHAR...		(44)9900-0088	
ALU_EMAIL	VARCHAR...		anafoz1987@4beat.com.br	
TUR_CODIGO	VARCHAR(6)		MUS1-N	
TIPOALU_CODIGO	SMALLINT			
Alu_foto	BLOB SUB...			

TUR_CODIGO	TUR_NOME
MUS1-N	Primeiro Período de Música Noturno
ADM1-N	Primerio Período de Administração Noturno
EEA1-D	Primerio Período de Engenharia Elétrica Diurno

Figura 5.77a - Carregando dados da tabela-mãe

A **Figura 5.77b** mostra a janelinha “Load Blod-Field From File” para localizar e carregar o arquivo da imagem desejada. Clique no botão **[Abrir]** para carregar efetivamente o arquivo...

O resultado é mostrado na tela da **Figura 5.77c**. Observe como fica o conteúdo do campo BLOB na janela de entrada de dados: apenas uns caracteres estranhos...

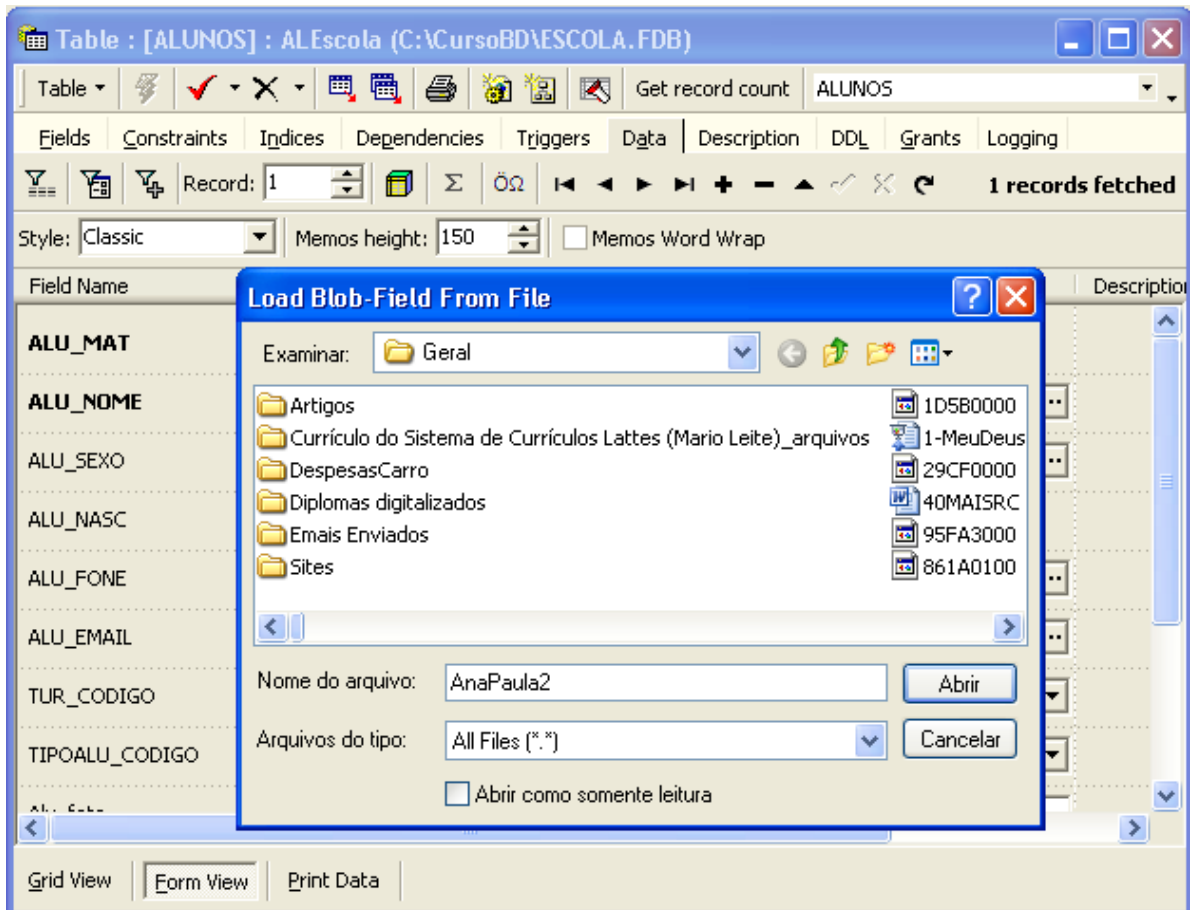


Figura 5.77b - Carregando o arquivo com a imagem desejada

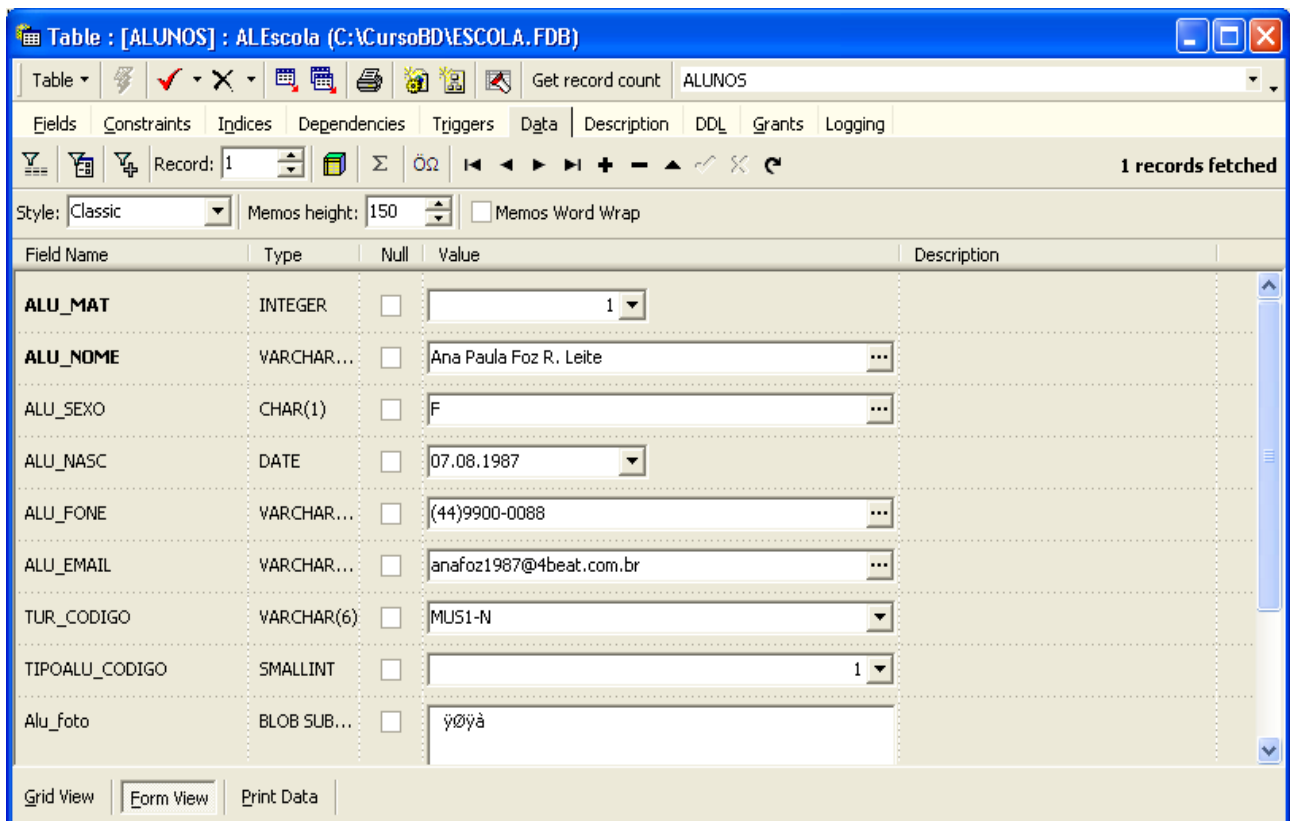
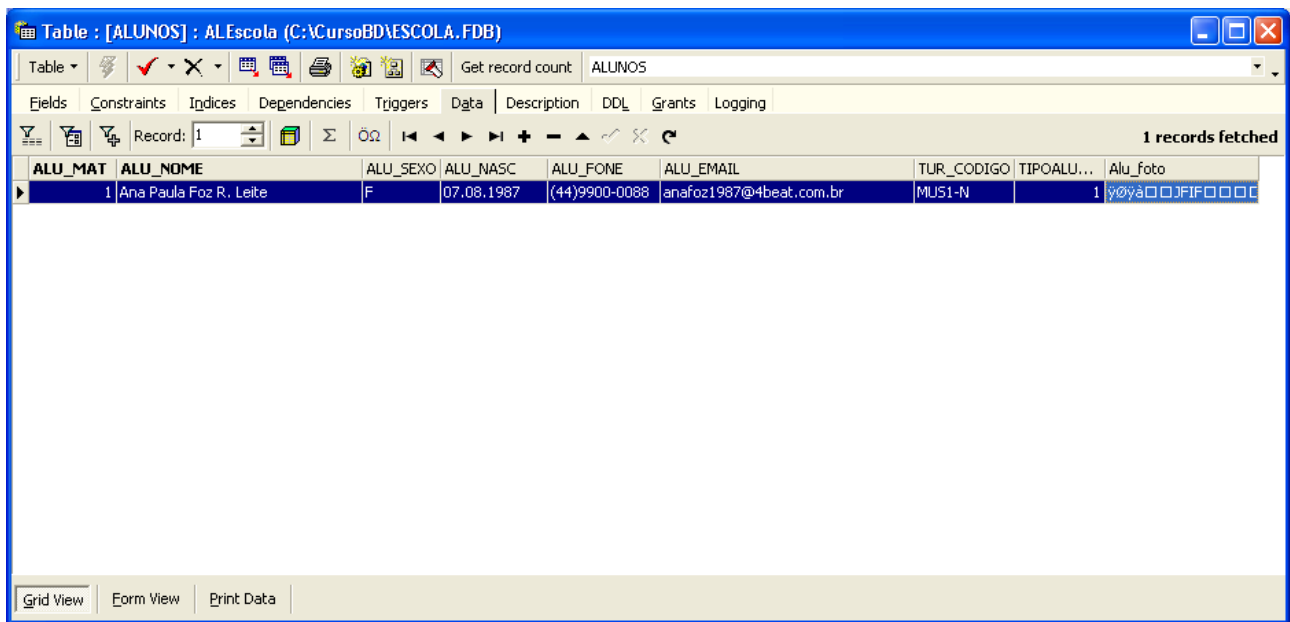


Figura 5.77c - O dado da imagem no campo BLOB

Para visualizar a imagem colocada no campo “Alu_foto” basta clicar na guia “Grid View” no rodapé da janela; o resultado é mostrado na **Figura 5.78a**: o registro com os dados do aluno.



The screenshot shows a database window titled 'Table : [ALUNOS] : ALEscola (C:\CursoBD\ESCOLA.FDB)'. The 'Data' tab is selected, showing a table with 1 record. The table has columns: ALU_MAT, ALU_NOME, ALU_SEXO, ALU_NASC, ALU_FONE, ALU_EMAIL, TUR_CODIGO, TIPOALU..., and Alu_foto. The record for 'Ana Paula Foz R. Leite' is displayed. At the bottom, the 'Grid View' button is highlighted.

ALU_MAT	ALU_NOME	ALU_SEXO	ALU_NASC	ALU_FONE	ALU_EMAIL	TUR_CODIGO	TIPOALU...	Alu_foto
1	Ana Paula Foz R. Leite	F	07.08.1987	(44)9900-0088	anafoz1987@4beat.com.br	MUS1-N	1	1 y@yâ□□JFIF□□□□

Figura 5.78a - Os dados do aluno “Ana Paula Foz. R. Leite

O segundo passo é dar um duplo clique sobre o campo BLOB; aparece a janela da **Figura 5.78b** onde é possível “ver” o conteúdo do campo BLOB em diversos tipos: como “texto”, “hexadecimal”, “imagem”, “formato RTF”, “página da Web” e “texto unicode”. E como estamos interessados em ver realmente a imagem, clique na guia “As Picture” (como imagem). Veja na **Figura 5.78c** o resultado; maravilha, né?!...

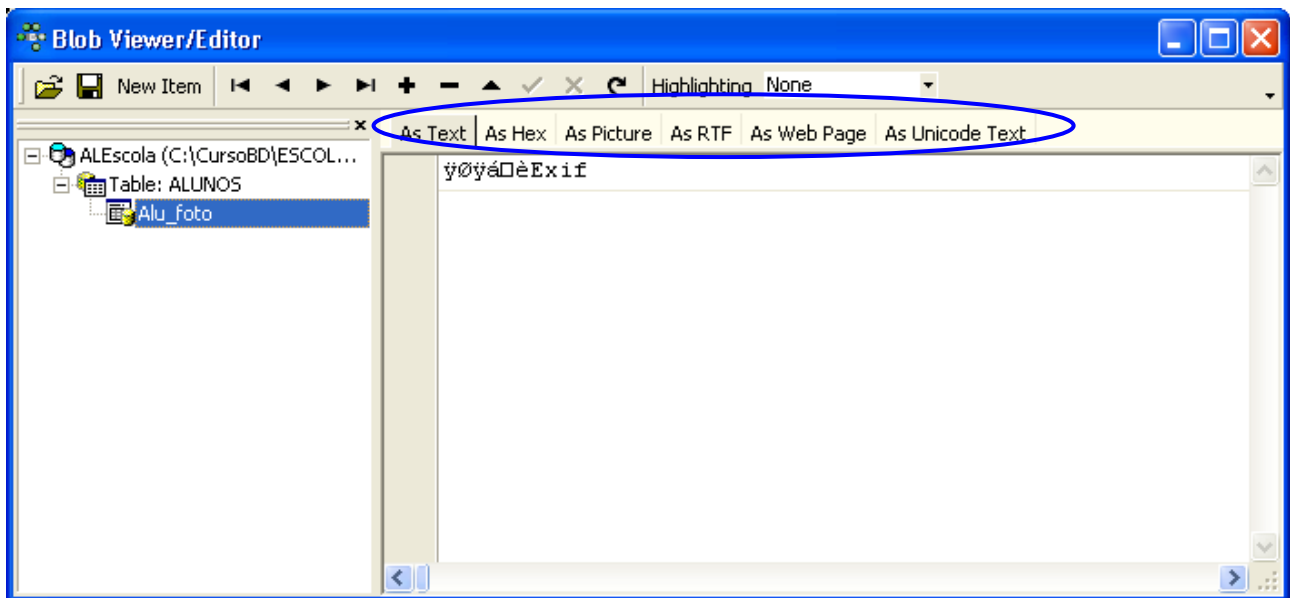


Figura 5.78b - As opções de visualização do conteúdo de um campo BLOB

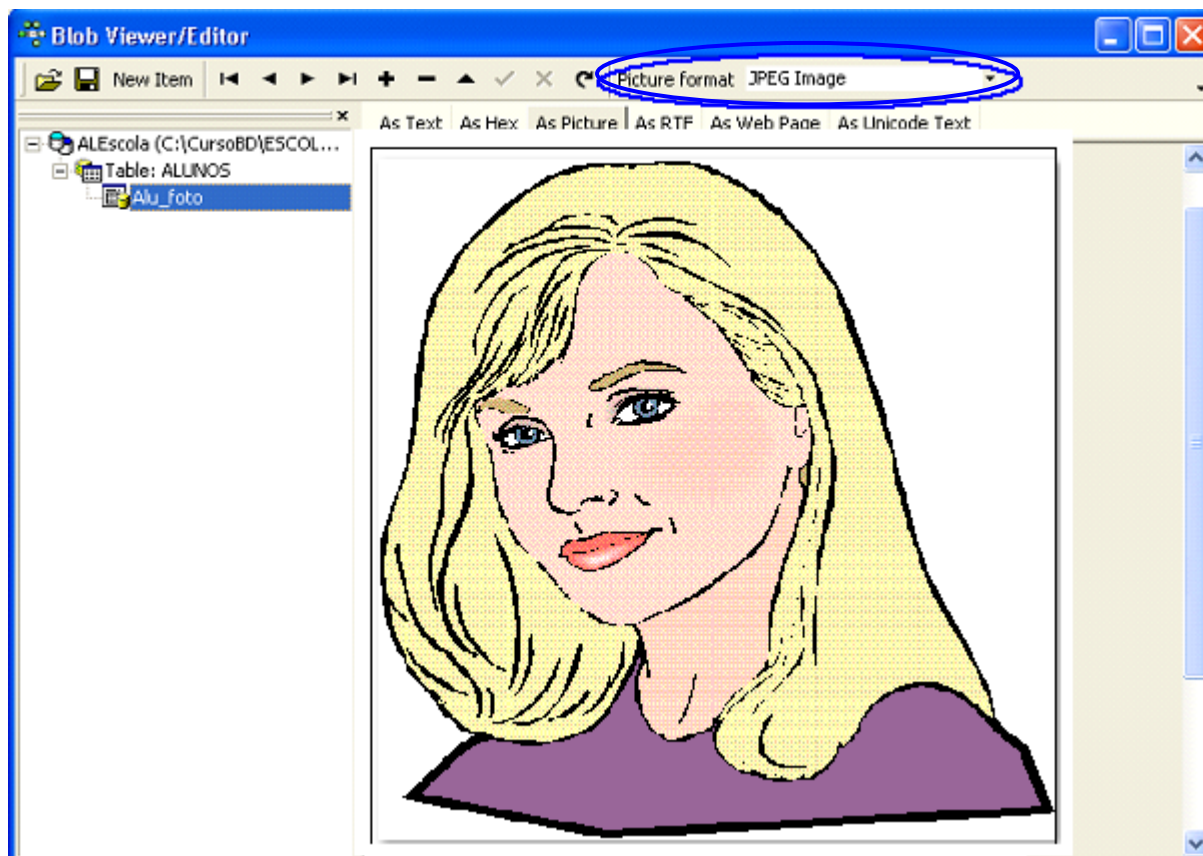


Figura 5.78c - O conteúdo do campo “Alu_foto” exibido como imagem


Obs) Note que na barra de ícones, na guia “Picture format”, foi selecionado **JPEG Image**; isto é importante para que a imagem apareça.

5.16 - Segurança no Firebird

5.16.1 - Criando novos usuários para o servidor

Normalmente, ao instalar Firebird, um usuário padrão é automaticamente definido para acessar os bancos do servidor: o *login* é **SYSDBA** (**SY**Stem **D**ata **B**ase **A**dministratortor - Administrador do Banco de Dados do Sistema) e a *password* é **masterkey** (chave mestra). Entretanto, é possível (e até recomendável) criar novos usuários para o servidor; para isto faça o seguinte:

1º) Na barra do *menu* principal clique em **Tools/User Manager**, e selecione a opção “Local” de servidor. Veja como fica na **Figura 5.79a**.

2º) Se o servidor não estiver conectado, conecte-o clicando no botão **[Connect]**, e na barra do *menu* principal clique em **Tools/User Manager** (ou clique diretamente no ícone ). Selecione a opção “Local” de servidor; veja como fica na **Figura 5.79b**. Nessa figura o servidor está sendo conectado com *login* e *password* padrões: SYSDBA e masterkey. Não é possível criar num novo usuário sem que o servidor esteja ativo.

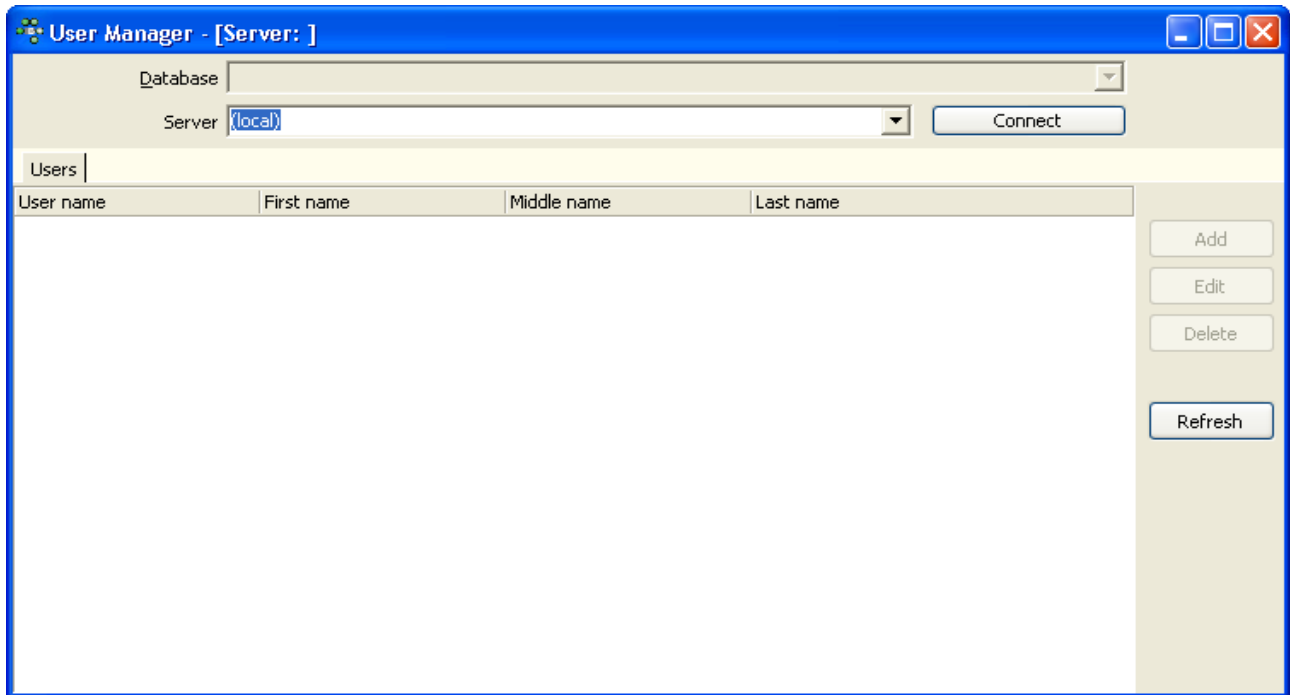


Figura 5.79a - Definido o tipo de servidor

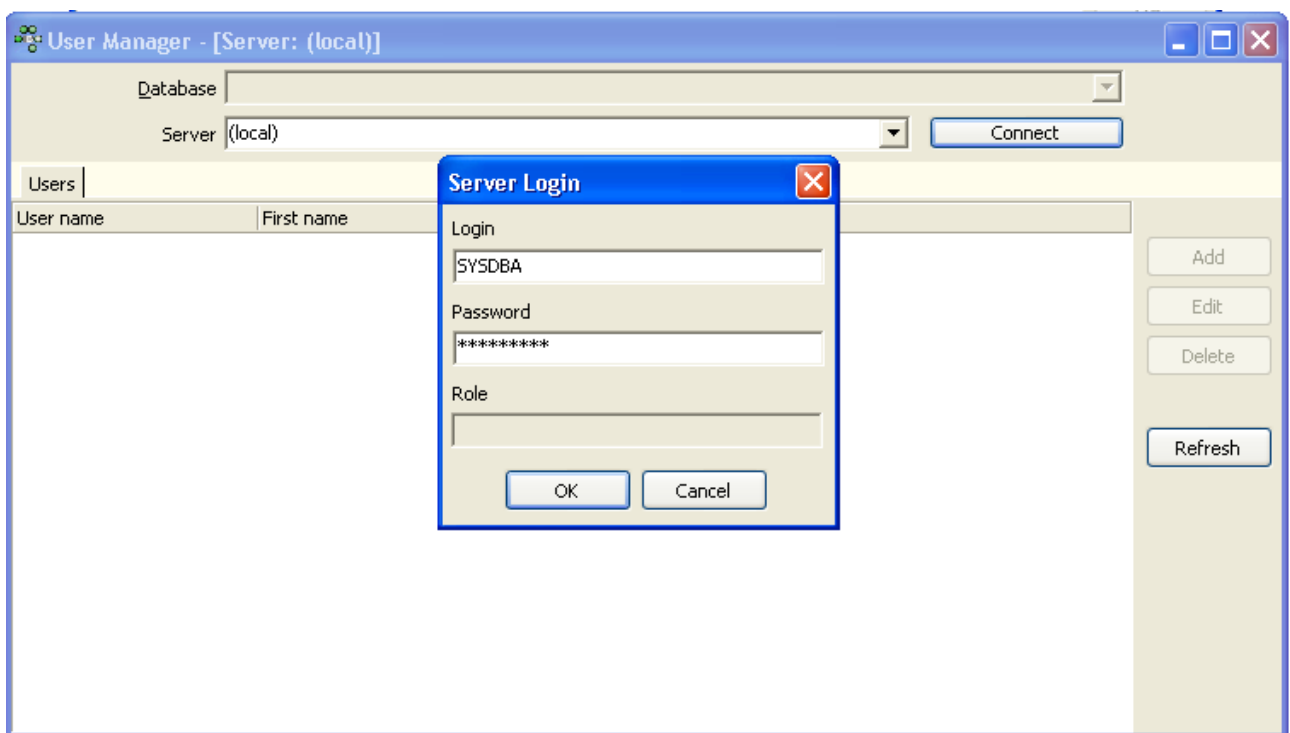


Figura 5.79b - Conectando o servido com *login* e *password* padrões

3º) Confirmando no botão **[OK]** da janelinha “Server Login” da **Figura 5.79b**, tela “User Manager” reaparece, agora mostrando todos os usuários atualmente registrados no servidor (confira na **Figura 5.79c**); que neste caso só existe até agora o usuário padrão SYSDBA.

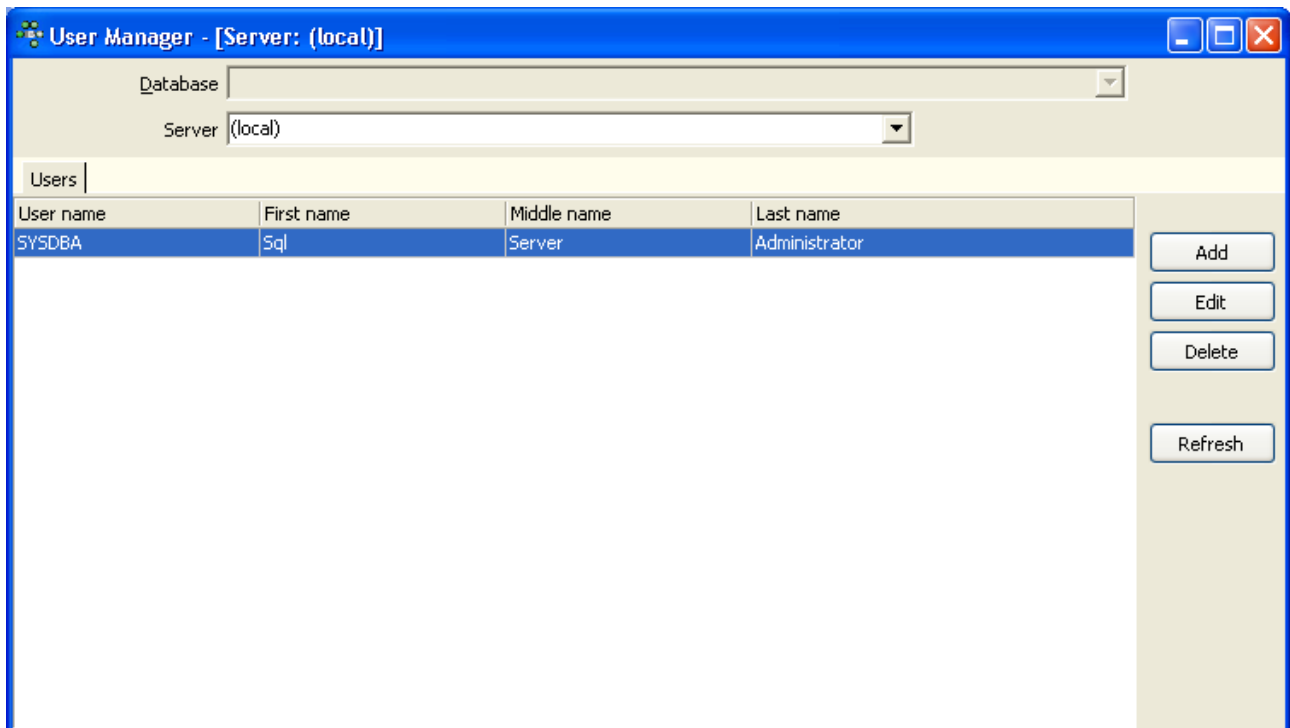


Figura 5.79c - Exibindo os usuários atuais *logados* no servidor

4º) Agora clique no botão **[Add]** para adicionar um novo usuário; as funcionalidades dos outros botões de ação são auto-explicativas. Preencha os dados do novo usuário solicitados na janela da **Figura 5.79d**.

The screenshot shows the 'New User' dialog box. It contains the following fields and controls:

- Name:** Text box containing 'MARLEITE'.
- Password:** Text box with masked characters '*****'.
- Confirm Password:** Text box with masked characters '*****'.
- System name:** Text box.
- Group name:** Text box.
- First Name:** Text box containing 'Mário'.
- Middle Name:** Text box containing 'Leite'.
- Last Name:** Text box.
- User ID:** Spin box set to '0'.
- Group ID:** Spin box set to '0'.
- Description:** Text area containing 'Administrador'.
- Buttons:** 'OK' and 'Cancel' buttons at the bottom right.

Figura 5.79d - Entrando com os dados do novo usuário do servidor

5º) Confirme no botão **[OK]** da janela da **Figura 5.79d**; novamente a janela “User Manager” reaparece, agora já mostrando que existe mais um usuário do servidor; observe na **Figura 5.79e**.

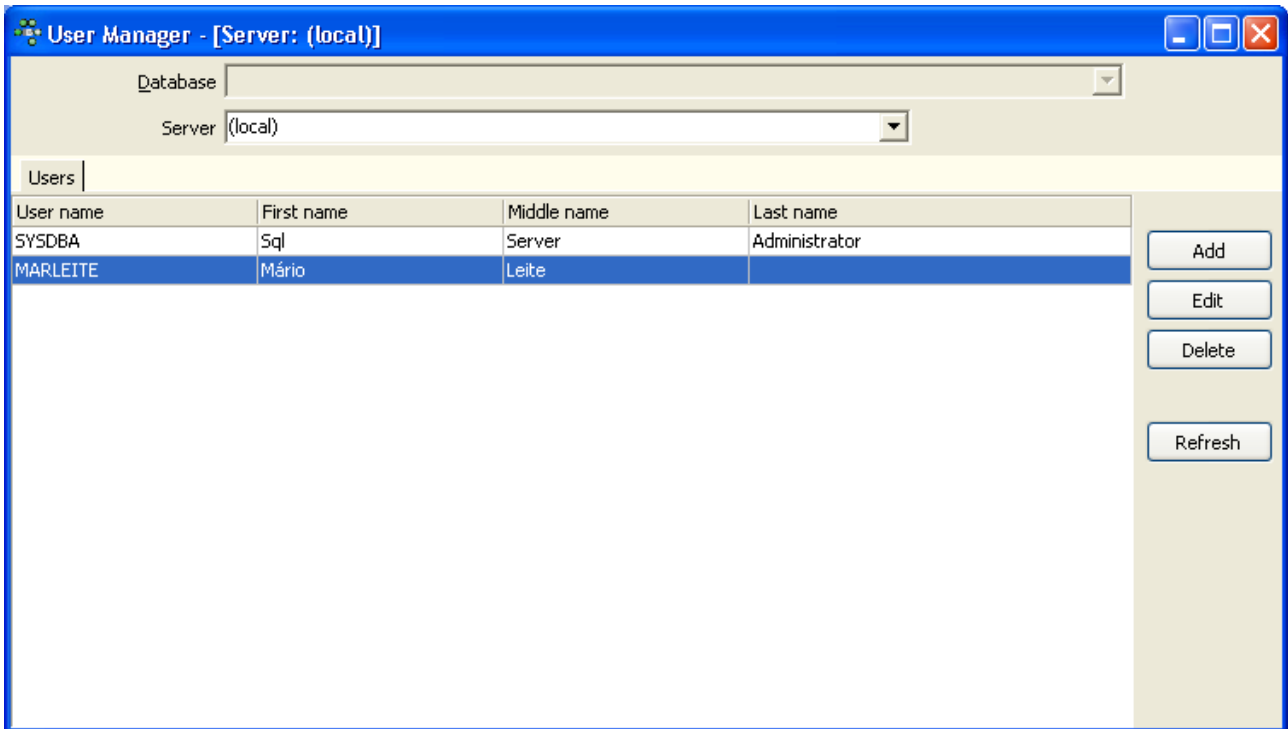


Figura 5.79e - Exibindo os usuários atuais do servidor (agora com um novo usuário)

Pronto; agora o novo usuário já pode *login* o banco com sua senha; veja o teste na **Figura 5.80a**.

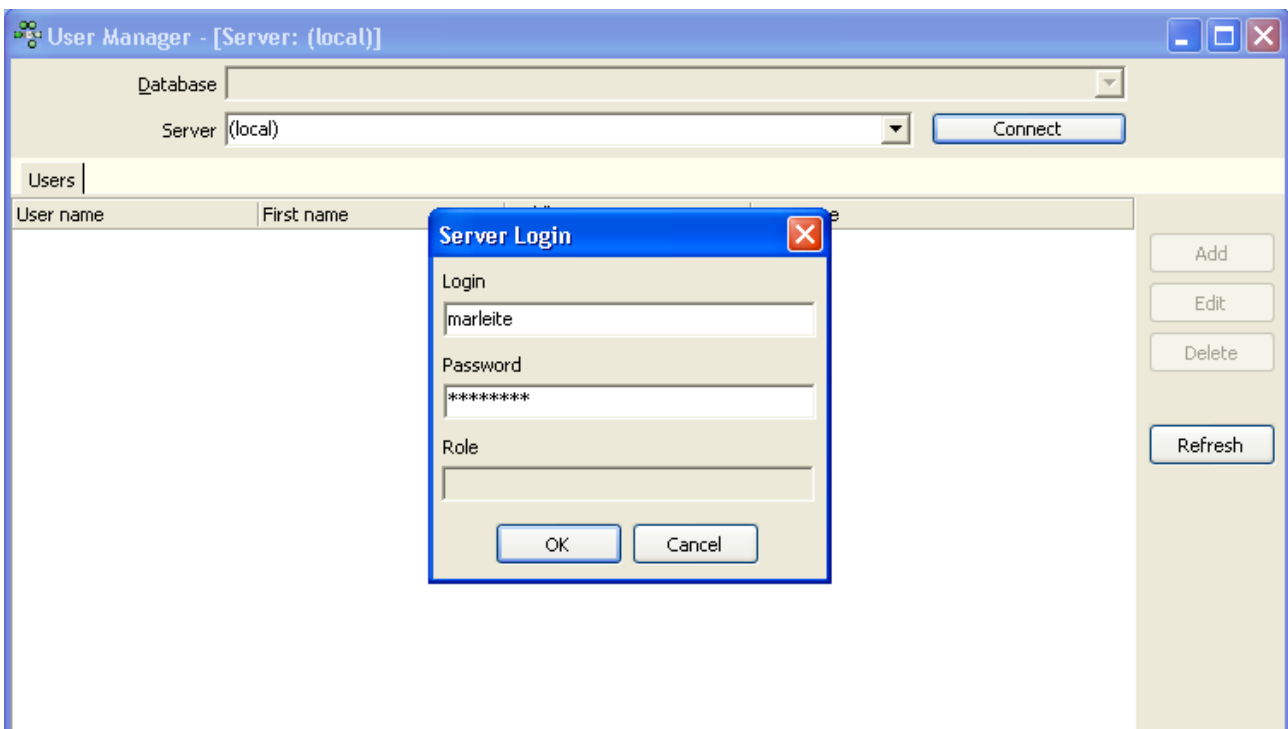


Figura 5.80a - O novo usuário fazendo *login* no servidor

Clicando no botão **[Refresh]** na janela da **Figura 5.79e**, note que agora o usuário *logado* no servidor é MARLEITE, como pode ser conferido na **Figura 5.80b**.

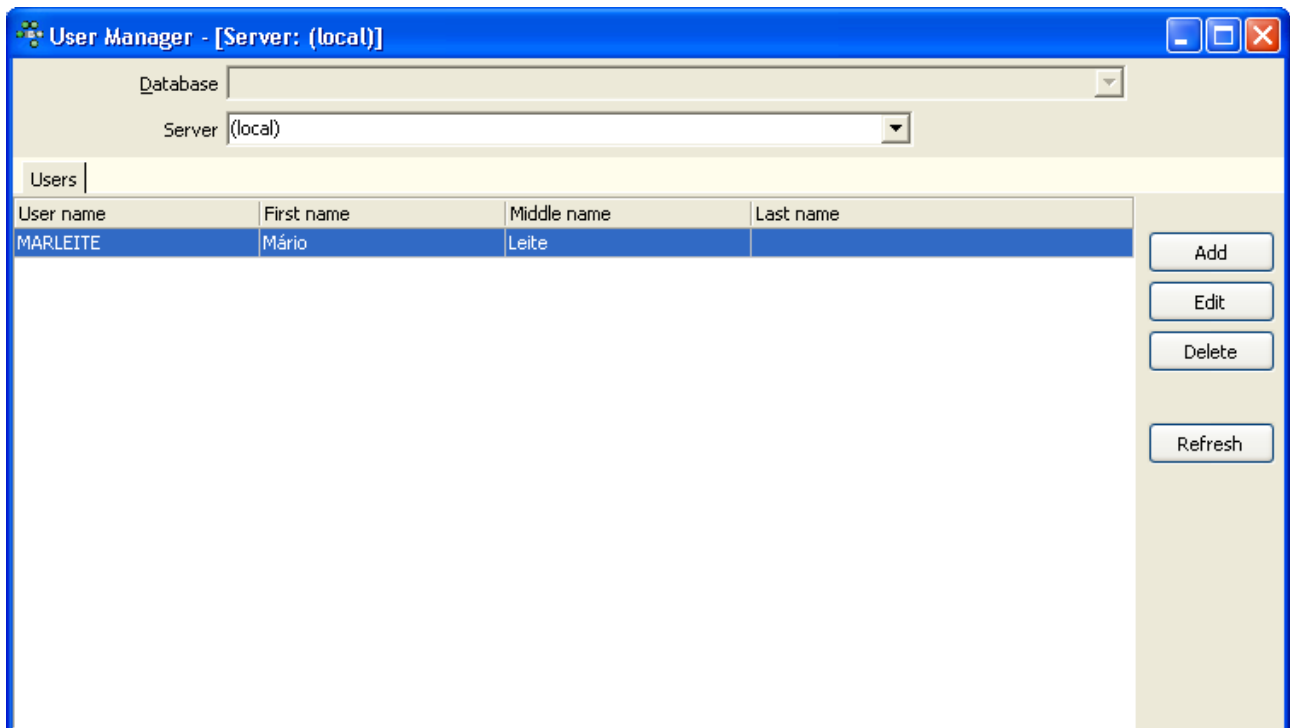


Figura 5.80b - O servidor *logado* pelo novo usuário

A **Figura 5.81** mostra o novo usuário registrando o banco “Escola.fdb”.

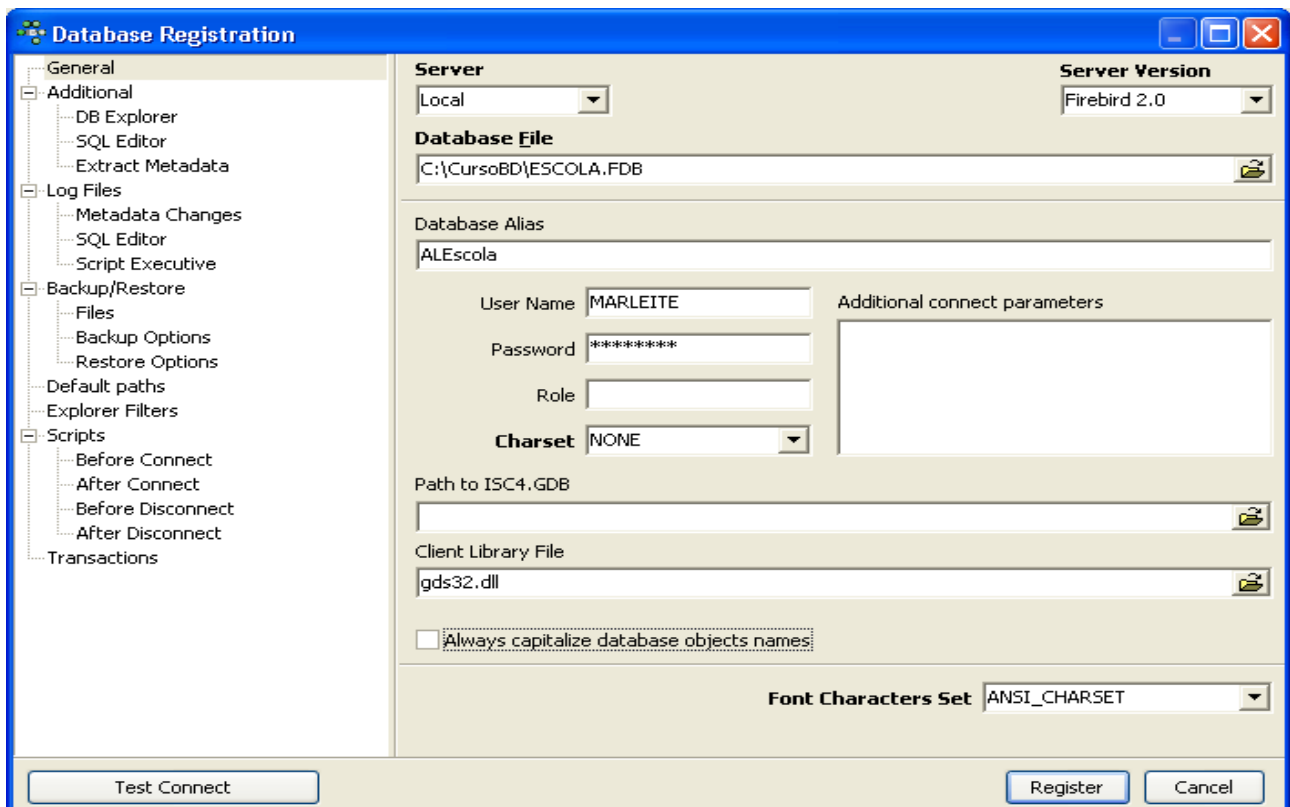


Figura 5.81 - O novo usuário registrando um banco no servidor

Confira na **Figura 5.82** que o banco “Escola.fdb” foi aberto normalmente com o novo usuário “MARLEITE”, sem problemas.

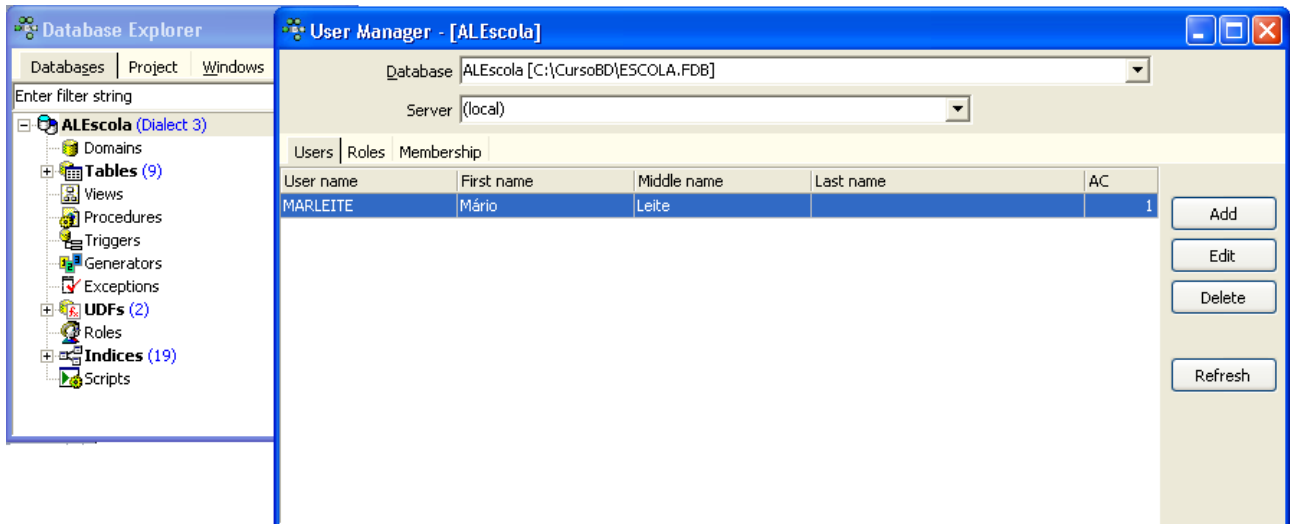


Figura 5.82 - O banco “Escola.fdb” registrado e conectado pelo novo usuário “MARLEITE”

5.16.2 - Alterando a senha padrão do servidor Firebird

Vamos agora tentar abrir o banco “Escola.fdb” como o *login* **SYSDBA** e *password* **masterkey**. Observe que isto sempre é, normalmente, possível, pois quando o servidor Firebird é instalado num computador o *login* é **SYSDBA** e a *password* **masterkey**; são os padrões definidos previamente pelo próprio servidor. Isto quer dizer que se você instalar o Firebird no seu computador pessoal mantendo esses padrões de *login* e senha, qualquer pessoa poderá acessar seus dados sem nenhum problema; é importante ter ciência disso! E como fazer para assegurar que só VOCÊ poderá acessar seus dados?! A solução é mudar a senha do servidor, passando da conhecida *masterkey* para uma outra qualquer que só você conheça. E para alterar a senha do usuário SYSDBA é relativamente fácil.

1º) Vá até **C:\Arquivos de programas(x86)\Firebird\Firebird_2_1\bin**; veja a Figura 5.83a;

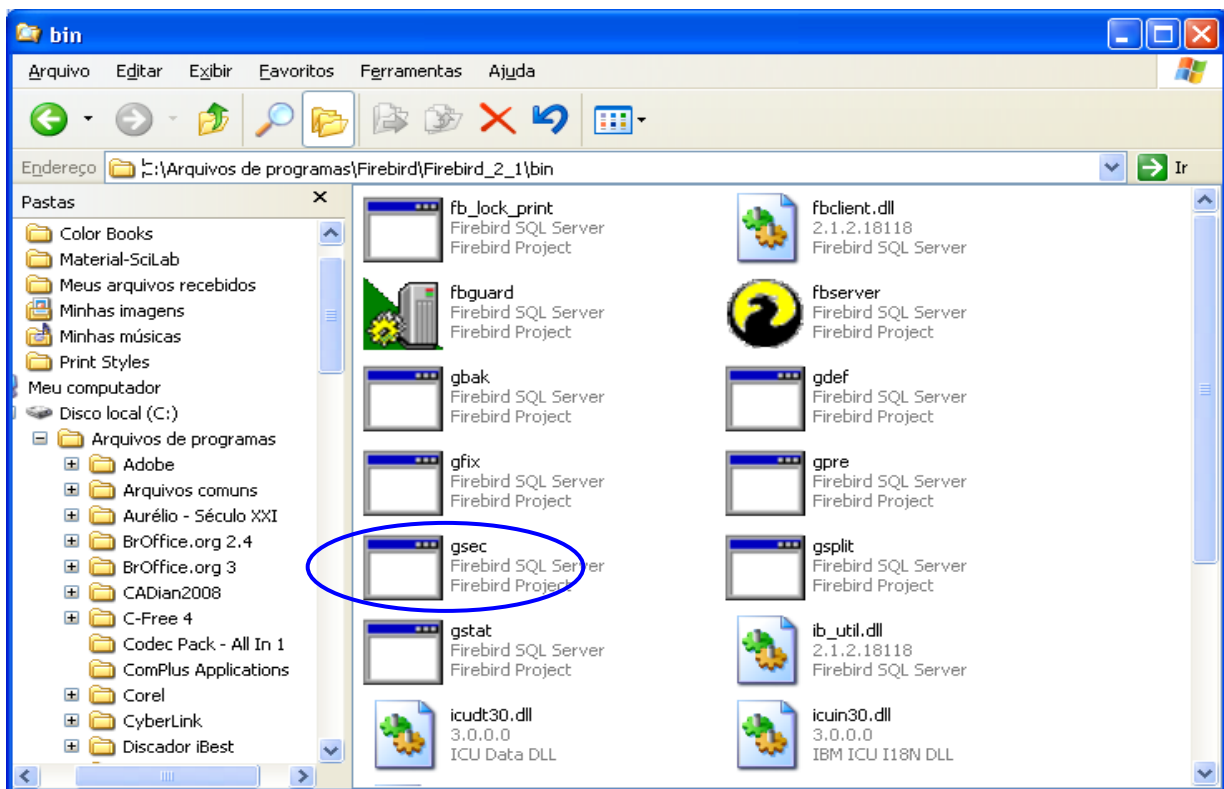


Figura 5.83a - Localizando o arquivo “gsec”

2º) Execute o programa **gsec**; uma janela DOS será aberta. Digite a seguinte linha de comando abaixo e confirme com <Enter>:

```
modify sysdba -pw imexivel@ <Enter>
```

Onde **imexivel** é a nova senha do usuário sysdba. Observe na **Figura 5.83b** como fica.



Figura 5.83b - Definindo a nova senha para o usuário SYSDBA

3º) Pressione a tecla **Enter** para executar a instrução.

Pronto; agora o usuário *sysdba* só poderá abrir bancos de dados no servidor Firebird com a senha **imexivel** e não mais com a tradicional *masterkey*. E como só você sabe disso, ninguém mais poderá abrir seus bancos de dados nesse servidor. Se o intruso tentar, o servidor gerará uma mensagem de erro numa janelinha como a da **Figura 5.84**, informando que não houve sucesso na tentativa de conectar o banco de dados no servidor.

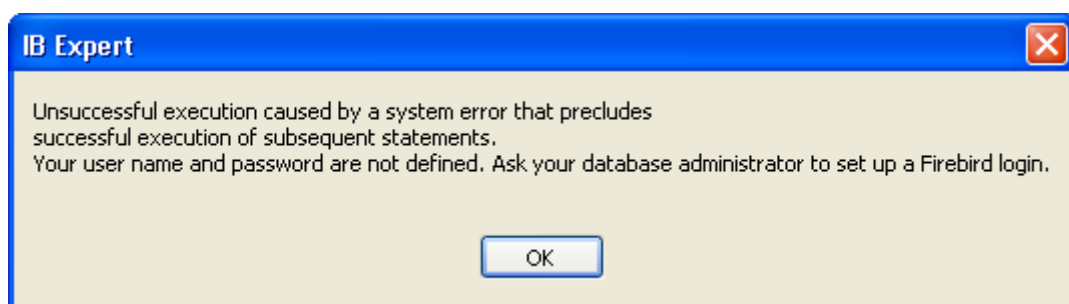


Figura 5.84 - Mensagem e erro na tentativa de registrar um banco sem autorização

Roteiro de Práticas

Capítulo 1

Atividade 1.1

Pesquise (em livros e/ou na Internet) e faça um resumo sobre o emprego de TI's em uma determinada empresa brasileira, relatando os resultados práticos obtidos.

Atividade 1.2

Pesquise (em livros e/ou Internet) e faça um resumo sobre os tipos de sistemas: abertos e fechados, apontando as vantagens e desvantagens de um em relação ao outro.

Atividade 1.3

Com relação á Atividade 1.2, qual dos dois tipos de sistemas se aplica a uma empresa? Explique a sua resposta!

Atividade 1.4

De acordo com a enciclopédia eletrônica “Wikipédia”, *dado e informação “são elementos intercambiáveis”* no contexto de um Sistema de Informações, mas com significados bem diferentes: informação é o resultado do processamento dos dados. Entretanto, na prática, muitas vezes uma informação passa a ser um dado. Mostre um exemplo em que isto pode acontecer.

Atividade 1.5

Qual o elemento que você acha mais importante num Sistema de Informações? Explique a sua escolha.

Atividade 1.6

A padaria do senhor “Manoel Nunes Pereira” tem apenas 5 empregados: ele próprio como gerente, sua esposa como caixa, seu filho mas velho como padeiro, e suas duas filhas como balconistas. É uma típica micro-empresa. Será que essa padaria necessita de um Sistema de Informações? Explique!

Atividade 1.7

Explique por que o *feedback* (realimentação) é tão importante num sistema aberto.

Atividade 2.1

Pesquise (em livros e/ou na Internet) e faça um resumo sobre 8 (oito) bancos de dados.

Atividade 2.2

Explique, com suas próprias palavras, as vantagens dos bancos de dados sobre os arquivos comuns.

Atividade 2.3

Considere dois conjuntos de elementos: “Fornecedor” e “Produtos”. Faça algumas associações entre esses dois conjuntos.

Atividade 2.4

Crie um MER relativo a um banco de dados de uma determinada clínica, contendo as tabelas abaixo. Os abritubos sublinhados são chaves primárias, e as interrogações são possíveis campos que podem existir ou não, dependendo de sua análise:

- **Medico**(Med_crm, Med_nome, Med_fone, ?)
- **Paciente**(Pac_codigo, Pac_sexo, Pac_nome, ?)
- **Especialidade**(Esp_codigo, Esp_nome, ?)
- **Quarto**(Qua_num, Qua_nome, ?)
- **Ala**(Ala_codigo, Ala_nome, ?)

Restrições:

Uma ala da clinica pode conter muitos quartos.

Um médico pode ter apenas uma especialidade.

Atividade 2.5

Com relação á Atividade 2.4, mostre quais são as tabelas fortes, as tabelas fracas e as tabelas associativas?. Explique a sua resposta.

Atividade 2.6

Marque com **V** as afirmativas verdadeiras e com **F** as afirmativas falsas.

- () Todas as tabelas que possuírem chave estrangeira são fracas no contexto.
- () Se um tabela não possuir chave estrangeira então ela é forte no contexto.
- () Todo tipo de relacionamento define uma tabela forte e uma fraca, distintas.
- () Se uma tabela tem duas chaves estrangeiras, então ela é associativa.
- () Todas as tabelas que possuírem chave primária são fortes no contexto.
- () O valor de um campo num registro não pode se repetir em outro registro.
- () Uma tabela pode possuir várias chaves secundárias.

Capítulo 3

Atividade 3.1

Pesquise (em livros e/ou na Internet) e faça um resumo sobre a forma normal de Boyce-Codd, mostrando um exemplo prático.

Atividade 3.2

Faça um resumo, com suas próprias, sobre Normalização e seu objetivo. Dê um exemplo de situação em que o banco de dados (ou uma simples tabela) não estando normalizado, pode trazer problemas na hora de extrair informações.

Atividade 3.3

Pesquise (em livros e/ou na Internet) e mostre um exemplo prático de normalização para a 5FN.

Atividade 3.4

Considere duas tabelas: “Vendedor” e “Clientes”, mostradas abaixo. Sabendo que um vendedor pode ter vários clientes, mas um cliente pertence a um único vendedor (considerando clientes fiéis!), responda: as duas tabelas estão normalizadas? Explique!

- **Vendedor**(Vendo_matricula, Vendo_nome, Vendo_endereco, Vendo_celular)
- **Clientes**(Cli_codigo, Cli_nome, Cli_fone, Vendo_matricula, Vendo_celular)

Atividade 3.5

Suponha um caso em que nas compras registradas por um fornecedor, para cada nota fiscal temos vários produtos com preços diferentes e quantidades compradas. Observe o esquema abaixo e crie um banco de dados para ele, normalizado até a 3FN.

Nota Fiscal:	12345
Data da compra:	28/09/09
Total da Nota:	R\$ 6.168,55

Produto	Preço	Quantidade
P123-0	35,12	15
P243_1	42,57	20
P315-0	22,65	15
P222-1	54,33	20
P542-0	67,28	50

Atividade 3.6

Explique, com suas próprias palavras, por que uma relação que possua chave primária simples já pode ser considerada normalizada na 2FN.

Capítulo 4

Atividade 4.1

Explique, com suas próprias palavras, a vantagem de uma linguagem de Quarta Geração (4GL) sobre uma linguagem 3GL.

Atividade 4.2

O que será apresentado quando for executada a instrução sql abaixo?

```
SELECT * FROM Produto WHERE(prod_codigo>5 AND prod_preço<50)ORDER BY prod_nome;
```

Atividade 4.3

Considerando a Atividade 3.4, escreva uma instrução sql que liste a matricula do vendedor, seu nome e o nome de todos os seus clientes.

Atividade 4.4

Considerando a Atividade 2.4, crie o banco de dados através de instruções sql.

Atividade 4.5

Considere três tabelas: “Clientes”, “Vendas” e “Pedidos”, onde um cliente pode ter várias vendas e também vários pedidos:

- **Clientes**(Cli_codigo, Cli_nome, Cli_fone)
- **Vendas**(Venda_num, Venda_data, Venda_valor, ?)
- **Pedidos**(Ped_num, Ped_data, ?)

Escreva duas instruções SQL: uma que liste os nomes dos clientes que fizeram compras antes da data atual (considere que a função agregada **GETDATE** dá a data atual) e outra que liste os nomes dos clientes que já fizeram algum pedido.

Atividade 4.6

Considerando a atividade anterior, escreva uma instrução sql atualize os valores das vendas em 2%, porém para valores acima de \$300,00.

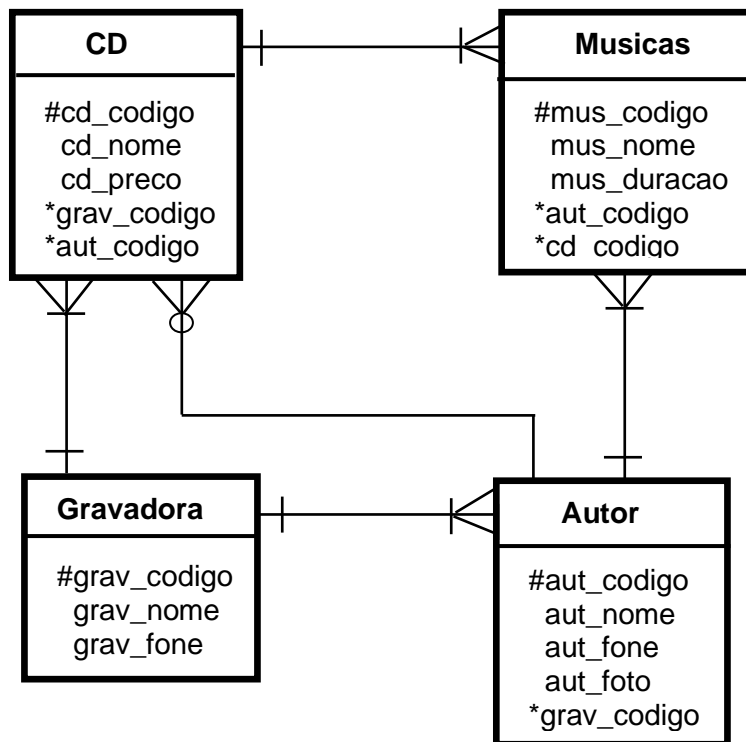
Atividade 4.7

Suponha que você tenha uma tabela que armazena informações sobre alunos de um colégio: Alunos(Alu_matricula, Alu_sexo, Alu_nota1, Alu_nota2, Alu_perodo).

Escreva uma instrução sql para listar a maior e a menor média dos alunos, porém apenas os períodos com mais de 10 alunos.

Atividade 4.8

Considerando os relacionamentos entre as tabelas abaixo, escreva duas instruções SQL: uma que imprima a quantidade de músicas por CD e outra que liste o preços médios de dos CD's, agrupado por gravadora



Atividade 4.9

Considerando à Atividade 4.8, explique o que resulta na execução da instrução sql:

```

SELECT CD.grav_codigo,Gravadora.grav_nome, AVG(cd_peco)
FROM CD,Gravadora
WHERE CD.grav_codigo=Gravadora.grav_codigo
GROUP BY CD.grav_codigo,Gravadora.grav_nome;
  
```

Atividade 4.10

Ainda com relação à Atividade 4.8, explique por que a instrução abaixo não é válida.

```

SELECT grav_codigo, AVG(cd_preco) FROM CD WHERE AVG(cd_preco)>20
GROUP BY grav_codigo
  
```

Atividade 4.11

Considerando o MER do banco "Locadora" (Figura 5.10 da apostila) escreva uma instrução sql para listar: título do filme em Português, nome do diretor e o gênero do filme. A listagem deve ser ordenada por título do filme, para cada diretor.

Atividade 4.12

Considerando o MER do banco "Locadora" (Figura 5.10 da apostila) escreva uma instrução sql que liste todos os filmes em que o ator "Arnold Schwarzenegger" não atuou como protagonista. A listagem deve conter, também, o gênero do filme e o diretor.

Atividade 5.1

Pesquise (em livros e/ou na Internet) sobre servidores de bancos de dados, e faça um resumo com suas próprias palavras. Dê exemplos de alguns servidores e suas principais características.

Atividade 5.2

Explique as vantagens de se utilizar uma ferramenta como o DBTools, IBExpert, Mysql-Front, pgAdmin, etc para criação e manutenção de elementos de bancos de dados.

Atividade 5.3

Qual a diferença entre *executar* e *commitar* uma instrução?

Atividade 5.4

Utilizando o IBExpert, crie um banco de dados relativo à **Atividade 4.8**. Insira pelo menos dez registros em cada tabela, com todos os campos preenchidos. Todos os campos “códigos” devem ser auto-incrementáveis e gerados por um “Generator”.

Atividade 5.5

Pesquise (em livros e/ou na Internet) e faça um resumo sobre “Roles” que aparece como um dos elementos de um servidor de bancos de dados. Apresente um exemplo prático.

Atividade 5.6

Com relação à **Atividade 4.8**, crie uma *stored procedure* que, recebendo o nome da música, retorne o nome do autor e da gravadora.

Atividade 5.7

Comiserando o MER da **Figura 5.10** (da apostila) crie uma *stored procedure* que dê o número de filmes já locados por um determinado cliente.

Atividade 5.8

Comiserando o MER da **Figura 5.70** (da apostila) crie uma *stored procedure* que mostre as médias obtidas nas avaliações (em ordem decrescente) com os respectivos nomes dos alunos.

Atividade 5.9

Utilizando o MS-Access, crie um banco dados para o MER da **Figura 5.10**, introduzindo uma nova tabela: “FilmesLocados”, que deve se relacionar com “Locacoes” e “Clientes”.

Bibliografia e Referências Bibliográficas

ALBERTIN, Alberto Luiz; ALBERTIN, Rosa Maria de Moura. *“Tecnologia de Informação e Desempenho Empresarial”*, Editora Atlas, São Paulo, 2005.

ANGELONI, Maria Terezinha; FERNANDES, Caroline Brito; SARTOR, Vicente Volnei de Santa; ROMANI, Cláudia e PEREIRA, Rita de Cássia de Faria. *“Gestão estratégica da informação e o processo decisório: uma preparação para a gestão do conhecimento”*, Florianópolis, ENEGEP, 1997.

CANTU, Carlos Henrique. *“Firebird 2.0 – O Banco de Dados do Novo Milênio”*, Editora Ciência Moderna, Rio de Janeiro, 2006.

LEITE, Mario. *“Acessando Bancos de Dados com Ferramentas RAD: Aplicações em Delphi”*, Editora Brasport, Rio de Janeiro, 2008.

LEITE, Mário. *“Programação Básica e Prática com Delphi”*. Editora LTC, Rio de Janeiro, 2005.

O'BREIN, James A. *“Sistemas de Informação e as Decisões Gerenciais na Era da Internet”*. Tradução da 9ª edição. Editora Saraiva, São Paulo, 2002.

POLONI, E. G. F. *“Administrando sistemas de informação”*. São Paulo: Futura, 2000.

TURBAN, Efrain; RAINER JR, R. Kelly; POTTER, Richard E. . *“Administração de Tecnologia da Informação - Teoria e Prática”*. Editora Campus, Rio de Janeiro, 2003

WALTON, Richard E. *“Tecnologia de Informação: o uso de TI pelas empresas que obtêm vantagem competitiva”*, Editora Atlas, São Paulo, 1993.