

Curso SQL

Por Rafael Goulart

Versão 1.0 – maio/2006

Índice

Introdução.....	3
Organização do Curso.....	3
Agradecimentos.....	3
Referências Utilizadas.....	3
Conceitos Gerais de SQL.....	5
O que é SQL.....	5
SGDB - Sistema de Gerenciamento de Banco de Dados.....	5
Bancos de Dados Relacionais.....	6
Tabelas.....	6
Registros (ou tupla).....	6
Colunas (Atributos ou Campos).....	6
Domínio (Tipos de Dados).....	7
Chave.....	7
Índices.....	7
Relacionamentos.....	7
A Linguagem SQL - Tipos de Palavras-Chave.....	8
DDL (Data Definition Language) - Linguagem de Definição de Dados.....	8
DML (Data Manipulation Language) - Linguagem de Manipulação de Dados.....	8
DCL (Data Control Language) - Linguagem de Controle de Dados.....	9
SQL em FIREBIRD - Introdução.....	10
História do FIREBIRD.....	10
Obtendo e Instalando FIREBIRD e Ferramentas.....	10
IBOConsole - comandos básicos.....	10
Registrando um Servidor.....	10
Registrando um Servidor Local (localhost).....	11
Registrando um Servidor Remoto.....	12
Criando/Registrando um Banco de Dados no Servidor.....	12
Criando um Banco de Dados no Servidor.....	14
Registrando um Banco de Dados no Servidor.....	15
Tela de Acesso ao Banco.....	16
Manutenção de Usuários.....	17
Manutenção de Tabelas.....	18
Executando comandos SQL.....	21
SQL Básico.....	23
Alguns padrões de sintaxe.....	23
SQL DDL Básico (Manipulando Objetos do BD).....	24
Criando um Banco de Dados (CREATE DATABASE).....	24
Conectando a um Banco de Dados (CONNECT).....	24
Criando Tabelas (CREATE TABLE).....	24
Tipos de Dados (Domínios Padrão) do FIREBIRD.....	25
Sintaxe CREATE TABLE (Criar Tabela).....	25
Opções comuns para campos na criação de tabelas.....	26
Criando chaves primárias (PK - Primary Key).....	26

Criando chaves estrangeiras (FK - Foreign Key).....	26
Chaves Primárias Autonumeradas.....	27
Alterando TABELAS (ALTER TABLE).....	28
Excluindo TABELAS, BANCOS e outros objetos.....	29
SQL DML Básico (Manipulando Dados).....	29
Inserindo Dados (INSERT).....	29
Selecionando Dados (SELECT).....	30
Sintaxe básica.....	31
Condições.....	31
Filtrando dados.....	31
Relacionando Tabelas.....	33
Ordenação (ORDER BY).....	34
Campos calculados, concatenação e funções.....	34
Agrupamento (GROUP e HAVING).....	34
Atualizando Dados (UPDATE).....	34
Excluindo Dados (DELETE).....	36
SQL DCL Básico (Manipulando Usuários e Permissões).....	38
Tipos de Privilégios.....	38
Permitindo acesso.....	38
Revogando acesso.....	38
Exemplo 01 - DEVA.....	39
Diagrama de Classes UML.....	39
Criação das Tabelas do Banco de Dados (DDL).....	40
Inserção de Dados nas tabelas.....	43

Introdução

Este curso teve inspiração na necessidade dar uma base para meus colegas do Curso de Administração com Habilitação em Análise de Sistemas, da [FASB - Faculdade São Francisco de Barreiras](#), em Barreiras/BA.

Temos a disciplina de Análise de Projetos e Sistemas e a disciplina Linguagem de Programação, mas nenhuma disciplina específica de Banco de Dados. Então, com o apoio dos professores de ambas as disciplinas, e utilizando como base análises em [UML](#) feitas em sala de aula, e para posterior desenvolvimento em Delphi do banco criado, elaborei este pequeno curso básico de [SQL](#).

O intuito inicial é utilizar o [FIREBIRD](#) (sucessor [software livre](#) do [INTERBASE](#)) como base para o curso', com a intenção de ser genérico o suficiente para que o conhecimento possa ser utilizado em outros bancos. Entretanto, num segundo momento, ele terá uma versão em [MySQL](#), e quem sabe me empolgo e faço também pra [Postgres](#)...

Em complemento aos conhecimentos adquiridos, utilizaremos análises feitas em [UML](#) em sala de aula, construindo as bases de exemplo através delas.

Acredito em retribuir o conhecimento que adquiri com tantos softwares e documentações livres através desta pequena colaboração. Também utilizarei fontes externas, principalmente para informações históricas, pois não tem sentido reinventar a roda...

Organização do Curso

Iniciamente a intenção era ter dois níveis, básico e intermediário. Mas percebi que o foco seria melhor se abordasse apenas o nível básico e o curso fosse “multibanco”. Por isso, a intenção é ter um conteúdo básico com uma explicação detalhada, facilitando o entendimento dos novatos, e deixar aprofundamentos para outro momento. Isto inclui comandos de criação e alteração de tabelas (**DDL**), manipulação e alteração de dados (**DML**) e controle de usuários e acesso (**DCL**).

Alguns recursos não serão abordados, a saber: **domains**, **stored procedures**, **triggers**, **exceptions**, **udfs**, **views**, por serem tópicos avançados e muito específicos de cada banco. Triggers serão utilizadas apenas no banco Firebird para alcançar resultados semelhantes ao de outros bancos em auto-numeração.

O [SQL](#) utilizado procurará ser o mais genérico possível, ou seja, não se deterá nos recursos específicos de cada banco. Lembre, este não é um curso de Firebird, MySQL ou Postgres, mas um curso de [SQL](#). Apenas serão identificadas algumas peculiaridades de cada banco para utilizar recursos de uso corriqueiro.

Agradecimentos

Em primeiro lugar agradeço ao Professor Fábio Callegari, que muito incentivou este trabalho cedendo suas aulas e acompanhando meu trabalho, e também Professor Alexandre Monge, que trabalhou suas aulas para construção dos exemplos que serão construídos neste curso.

E claro, à minha turma que me aturou.

Referências Utilizadas

[Wikipedia - A enciclopédia livre](#)

Vários trechos sobre SQL, história dos bancos e conceitos foram retirados da Wikipedia. Em

especial:

- [SQL](#)
- [UML](#)
- [Bancos de Dados Relacional](#)
- [Sistemas de Gerenciamento de Banco de Dados](#)

Várias apostilas foram consultadas, e duas fontes básicas foram: [SQL Magazine](#) [ClubDelphi](#)

Infelizmente muitas delas não possuem referência a autoria, possivelmente por serem cópias de outros autores desrespeitando a fonte. Não são citadas aqui, mas encontrando algum trecho que conheça e de que saiba o autor, entre em contato comigo para a devida referência.

Agradeço especialmente às apostilas dos seguintes autores:

- [Apostila de Banco de Dados e SQL - Prof. Jorge Surian & Prof. Luiz Nicochelli](#)
- [Banco de Dados - Prof. Renato Fileto](#)
- [Apostila de Firebird 1.0 - Autor: Anderson Haertel Rodrigues & Colaboração: Marcus Boi](#)

E claro, a documentação dos próprios bancos, encontrada em:

- [Firebird 1.5 Quick Start Guide](#) (em inglês)
- [PostgreSQL BR](#)
- [Manual de Referência do Mysql 4.1](#)

Ferramentas utilizadas para criação deste curso

- [Dia - Editor de Diagramas](#) - Para criação de diagramas de classes UML
- [Gimp](#) - Para editar as imagens utilizadas.

Conceitos Gerais de SQL

O que é SQL

[O texto deste tópico foi extraído da Wikipédia, a enciclopédia livre](#)

Structured Query Language, ou **Linguagem de Consulta Estruturada** ou **SQL**, é uma linguagem de pesquisa declarativa para banco de dados relacional (bases de dados relacionais). Muitas das características originais do SQL foram inspiradas na álgebra relacional.

SQL é normalmente pronunciado em português como “esse-quê-ele”, porém sua pronúncia correta deveria se “síquel”, do inglês “sequel”, ou “alguma coisa que segue outra coisa”. SQL é uma brincadeira com o nome da primeira linguagem de consulta QUEL.

Embora o SQL tenha sido originalmente criado pela IBM, rapidamente surgiram vários “dialectos” desenvolvidos por outros produtores. Essa expansão levou à necessidade de ser criado e adaptado um padrão para a linguagem. Esta tarefa foi realizada pela American National Standards Institute (ANSI) em 1986 e ISO em 1987.

O SQL foi revisto em 1992 e a esta versão foi dado o nome de SQL-92. Foi revisto novamente em 1999 e 2003 para se tornar SQL:1999 (SQL3) e SQL:2003, respectivamente. O SQL:1999 usa expressões regulares de emparelhamento, queries recursivas e gatilhos (triggers). Também foi feita uma adição controversa de tipos não-escalados e algumas características de orientação a objeto. O SQL:2003 introduz características relacionadas ao XML, sequências padronizadas e colunas com valores de auto-generalização (inclusive colunas-identidade).

Tal como dito anteriormente, o SQL, embora padronizado 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.

Outra aproximação é permitir para código de idioma processual ser embutido e interagir 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, ou C, entre outras linguagens.

SGDB - Sistema de Gerenciamento de Banco de Dados

De uma forma beeeem simplificada, SGDB é um software responsável pelo gerenciamento (armazenamento e recuperação) dos dados no Banco de Dados. Mas... existe muita discussão sobre o que pode ser considerado SGDB ou simplesmente um GA - Gerenciador de Arquivos.

Não é do escopo deste curso entrar nestas discussões. Você pode encontrar mais detalhamento teórico e visões sobre este assunto nestas duas referências:

- [Apostila de Banco de Dados e SQL - Prof. Jorge Surian & Prof. Luiz Nicochelli](#)
- [Banco de Dados - Prof. Renato Fileto](#)

O que seria interessante considerar é que a depender a referência e do autor, o MS Access pode ou não ser considerado um SGDB, assim como o MySQL, pois nenhum dos dois possui todas as características “básicas” de um SGDB. Mas... o MySQL, em sua versão 5.0 os possui, e em versões anteriores possuía várias destas características através de motores adicionais como o INNODB. Já o MS Access, que muitos autores colocam como SGDB, não possui um servidor dedicado, sendo seus arquivos manipulados diretamente pelos clientes através da biblioteca JET... Ou seja, é muita

discussão.

IMHO - Em minha humilde opinião - apesar de possuir muitos recursos interessantes, o MS Access não pode ser comparado ao MySQL, mas também não pode ser comparado ao Dbase... E entendo também que hoje o foco está muito mais na ampla disseminação do MySQL e seu foco em desempenho, o que incomoda bancos mais “completos” Oracle, Interbase/Firebird, Postgres, DB2, Ingress, Progress, que perdem mercado para um concorrente “incompleto”.

É fato que o Postgres e o Firebird, apenas citando opções livres, são mais completos e complexos que o MySQL, e até por isto menos populares entre programadores novos. Mas têm sua força.

Bancos de Dados Relacionais

Existem vários modelos de bancos de dados tais quais:

- **Modelo Orientado ao Registro** - *Modelo Relacional, Modelo Hierárquico e Modelo em Rede*;
- **Modelo Semântico**: *Modelo Entidade-Relacionamento e o Funcional*;
- **Modelo Orientado ao Objeto**: *Modelos O2 e o de Representação de Objetos*.

(Obs.: veja em [Referências Utilizadas](#) maiores aprofundamentos sobre a questão teórica).

O FIREBIRD, MySQL e POSTGRES utilizam o **Modelo Relacional**, que é o modelo mais largamente utilizado pelos SGBDs. Alguns conceitos deste modelo são importantes para a continuidade de nossos estudos:

(O trecho a seguir foi adaptado da [Wikipedia](#))

Tabelas

Todos os dados de um banco de dados relacional (BDR) são armazenados em tabelas. Uma tabela é uma simples estrutura de linhas e colunas. Cada linha contém um mesmo conjunto de colunas mas as linhas não seguem qualquer tipo de ordem. Em um banco/base de dados podem existir uma ou centenas de tabelas. O limitador é imposto exclusivamente pela ferramenta de software utilizada.

	CPF	Nome
Registro 1	152.487.265-42	João da Silva
Registro 2	687.481.682-88	Maria Joaquina
Registro 3	001.645.852.12	Agnaldo Souza

Registros (ou tupla)

Cada linha, formada por uma lista ordenada de colunas, representa um registro (ou tupla). Os registros não precisam necessariamente conter dados em todas as colunas, os seus valores podem ser nulos.

Formalmente falando, uma tupla é uma lista ordenada de valores, onde cada valor é do domínio especificado pelo atributo definido no esquema de relação.

Colunas (Atributos ou Campos)

As colunas de uma tabela, são também chamadas de Atributos ou Campos. Cada atributo/campo pertence a um domínio (tipo de campo), que define os valores que podem ser associados aquele

atributo.

Domínio (Tipos de Dados)

Os Domínios possuem características que definem os possíveis valores que serão armazenado em um atributo de uma tupla. Por exemplo. Em um campo do tipo numérico, serão somente armazenados números. Os sistemas de banco de dados possuem regras para consistir os dados que são armazenados.

Chave

As tabelas relacionam-se umas as outras através de chaves. Uma chave é um conjunto de um ou mais campos que determinam a unicidade de cada registro.

Por exemplo, se um banco (base) de dados tem como chave Código do Produto + ID Sistema, sempre que acontecer uma inserção de dados, o sistema de gerenciamento de banco (base) de dados irá fazer uma consulta para identificar se o registro não se encontra gravado na tabela. Caso exista um novo registro não será criado, sendo que apenas a alteração do registro existente será possível. A unicidade dos registros, determinada por sua chave, também é fundamental para a criação dos índices.

Temos dois tipos de chaves:

- **Chave Primária: (PK - Primary Key)** é a chave que identifica cada registro dando-lhe unicidade. A chave primária nunca se repetirá. A maioria dos bancos também exige que ela seja NÃO NULA (NOT NULL) além de única.
- **Chave Estrangeira: (FK - Foreign Key)** é uma chave formada pela chave primária de outra tabela e a chave de um campo da tabela que recebe o relacionamento. Define um relacionamento entre as tabelas e pode ocorrer repetidas vezes.

Observação: Geralmente Chaves Primárias são de apenas um campo, mas podem ser compostas, ou seja, vários campos fazem parte dela. Isto determina que a *combinação entre os campos* é que precisa ser única, e não os campos que isoladamente a compõe.

Índices

O limitador é imposto exclusivamente pela ferramenta de software utilizada. Sendo assim, para a recuperação dos dados é necessário a existência de mecanismos que facilitem a consulta, proporcionando uma performance aceitável para a mesma. Para isso, os sistemas de bancos de dados relacionais criam índices das tabelas, sendo que esses índices são atualizados constantemente.

Caso o índice se corrompa por algum motivo, é possível que pesquisas possam retornar resultados não desejados ou que inserções de chaves duplicadas aconteçam. Nesse caso o banco de dados será corrompido também. Os sistemas de bancos (bases) de dados possuem mecanismos de evitar que esses eventos ocorram como também possibilitam a recuperação dos índices e consistência da tabela caso eles ocorram.

Relacionamentos

Como o próprio nome sugere, um(a) BDR possui diversos relacionamentos entre as tabelas existentes.

Um relacionamento é feito ligando-se um campo de uma tabela X com um campo de uma tabela Y ou tabela estrangeira. Por exemplo, se pedidos (em uma tabela) estão relacionados a um cliente (em

outra tabela), o SGBD poderá bloquear a remoção deste cliente enquanto ele possuir pedidos registrados. (Observação: no caso de chaves primárias compostas, a relação se dará entre todos os campos desta chave)

Existem alguns tipos de relacionamentos possíveis:

- Um para um (1 para 1)
- Um para muitos (1 para N)
- Muitos para muitos (N para N), que não pode ser implementado diretamente no modelo relacional e tem que ser construído com uma tabela auxiliar.

A Linguagem SQL - Tipos de Palavras-Chave

(O trecho a seguir foi adaptado da [Wikipedia](#))

Alguns grupos de comandos do SQL são definidos pelas funções exercidas no banco de Dados. Existem algumas subdenominações, mas o mais comum é definir em dois grandes grupos:

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

Permite a criação e alteração de objetos do banco de dados. Apesar de baseado na SQL padrão, geralmente existem extensões para cada SGDB utilizado. Os comandos clássicos são:

- **CREATE** - cria um objeto (uma tabela, por exemplo) dentro do banco de dados (ou o próprio);
- **DROP** - apaga um objeto do banco de dados;
- **ALTER** - altera um objeto do banco de dados.

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

Permite a alteração dos dados dentro das tabelas. Os mais comuns são:

- **SELECT** (SELECIONAR) - para seleção de dados em um ou mais registros
- **INSERT** (INSERIR) - para inclusão um registro
- **UPDATE** (ATUALIZAR) - para modificação valores de um registro
- **DELETE** (EXCLUIR) - para exclusão de um registro
- **TRUNCATE** (TRUNCAR) - exclui todos registros da tabela

Alguns comandos são utilizados para manipular a TRANSAÇÃO. Transação é um conceito muito importante em SGDBs, pois permite demarcar um bloco de comandos que será executado ou não, mas sempre em conjunto. Se um comando falhar, todo o conjunto de comandos falha, ou então é possível testar isoladamente alterações e só efetivar no banco de dados as alterações quando se considera seguro. Os comandos utilizados nesta realidade são:

- **BEGIN TRANSACTION** (ou **START TRANSACTION**, ou **BEGIN WORK**, ou **BEGIN TRANS**, dependendo do dialeto SQL) pode ser usado para marcar o começo de uma transação de banco de dados que pode ser completada ou não (geralmente é utilizado um **END** para finalizar a transação).
- **COMMIT** envia todos os dados das mudanças permanentemente.
- **ROLLBACK** faz com que as mudanças nos dados existentes desde que o último **COMMIT** ou **ROLLBACK** sejam descartadas.

COMMIT e **ROLLBACK** interagem com áreas de controle como transação e locação. Ambos terminam qualquer transação aberta e liberam qualquer cadeado ligado a dados. Alguns bancos

podem trabalhar com AUTO-COMMIT, ou seja, confirmam qualquer comando após sua execução. Isto pode ser útil, mas retira uma camada de segurança importante em aplicações importantes.

Observação: algumas referências utilizam o termo DQL (Data Query Language)- Linguagem de Consulta de Dados para o comando **SELECT**, exclusivamente.

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

DCL controla os aspectos de autorização de dados e licenças de usuários para controlar quem tem acesso para ver ou manipular dados dentro do banco de dados.

Duas palavras-chaves da DCL:

- **GRANT** - autoriza ao usuário executar ou setar operações.
 - **REVOKE** - remove ou restringe a capacidade de um usuário de executar operações.
-

SQL em FIREBIRD - Introdução

História do FIREBIRD

[O texto deste tópico foi extraído da Wikipédia, a enciclopédia livre](#)

Firebird (algumas vezes chamado de FirebirdSQL) é um sistema de manutenção de bases de dados. Corre em Linux, Windows e uma variedade de plataformas Unix. A Fundação FirebirdSQL faz a manutenção e desenvolvimento do Firebird.

Baseado no código do InterBase da Borland, quando da abertura de seu código na versão 6.0 (em 25 de Julho de 2000), alguns programadores em associação, assumiram o projeto de identificar e corrigir inúmeros bugs da versão original, surgindo aí o Firebird, que se tornou um banco com características próprias, obtendo uma aceitação imediata no círculo de programadores, a primeira versão 1.0. Quase que totalmente ainda compatível com sua origem, estando atualmente em sua versão 1.5, com muitas novidades... A versão 2.0 em fase de produção ainda, deverá trazer muitas inovações, já está se falando até em uma versão 3.0 que hoje tem o codinome Vulcan, cujas características já então seria de um super banco de dados, seu maior diferencial ainda se baseia na gratuidade, o banco é free em todos os sentidos... não há limitações de uso, e seu suporte amplamente discutido em listas na internet, o que facilita enormemente a obtenção de ajuda técnica.

Obtendo e Instalando FIREBIRD e Ferramentas

Na data de confecção deste material (maio/2006), a última versão estável do Firebird era 1.5.3. Você sempre poderá obter a última versão no link [aqui](#). A versão utilizada neste curso é a SuperServer para Windows.

[Download do Firebird 1.5.3](#)

Como ferramenta de administração foi escolhido o IBOConsole, que é uma versão atualizada para Firebird do IBConsole, acompanhante das antigas versões do Interbase. Embora não seja o mais poderoso, é livre e estável. Você encontra a última versão [aqui](#).

[Download do IBOConsole 1.1.12](#)

Agora vamos passo a passo instalar cada um deles:

IBOConsole - comandos básicos

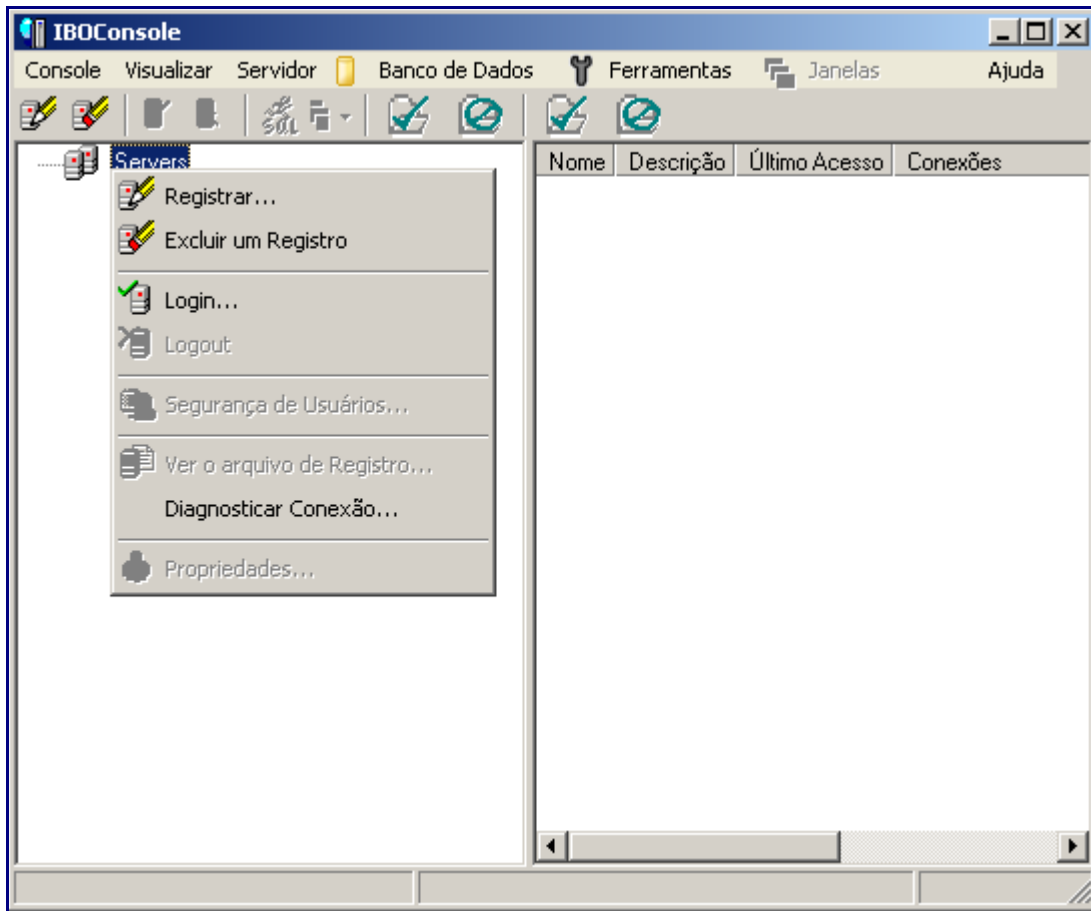
Para utilizar o IBOConsole, é preciso conhecer sua interface e aprender algumas operações básicas. Vamos lá.

Registrando um Servidor

O primeiro passo é registrar um servidor. Entende-se por um servidor um computador que hospeda um ou mais bancos de dados do Firebird. Para registrá-lo, é necessário saber seu endereço IP ou nome de rede (com exceção do servidor na máquina onde o IBOConsole está instalado, conhecido como local).

Utilize o menu **Servidor** → **Registrar**. ou clique com o botão direito em **Servers** e na opção

Registrar...



Registrando um Servidor Local (localhost)

A tela sugere inicialmente o registro do servidor residente na própria máquina, referenciado como **Local Server**. O superusuário do FIREBIRD (todo poderoso administrador do servidor) é o **SYSBDA**, com a senha padrão **masterkey**.

- Usuário: **SYSBDA**
- Senha: **masterkey**

Register Server and Connect

Informações do Servidor

☒ Local Server ☐ Remote Server

Nome do Servidor: Protocolo de Rede:

Nome do Alias:

Descrição:

☒ Salvar as informações do Alias

Informações do Login

Usuário:

Senha:

Registrando um Servidor Remoto

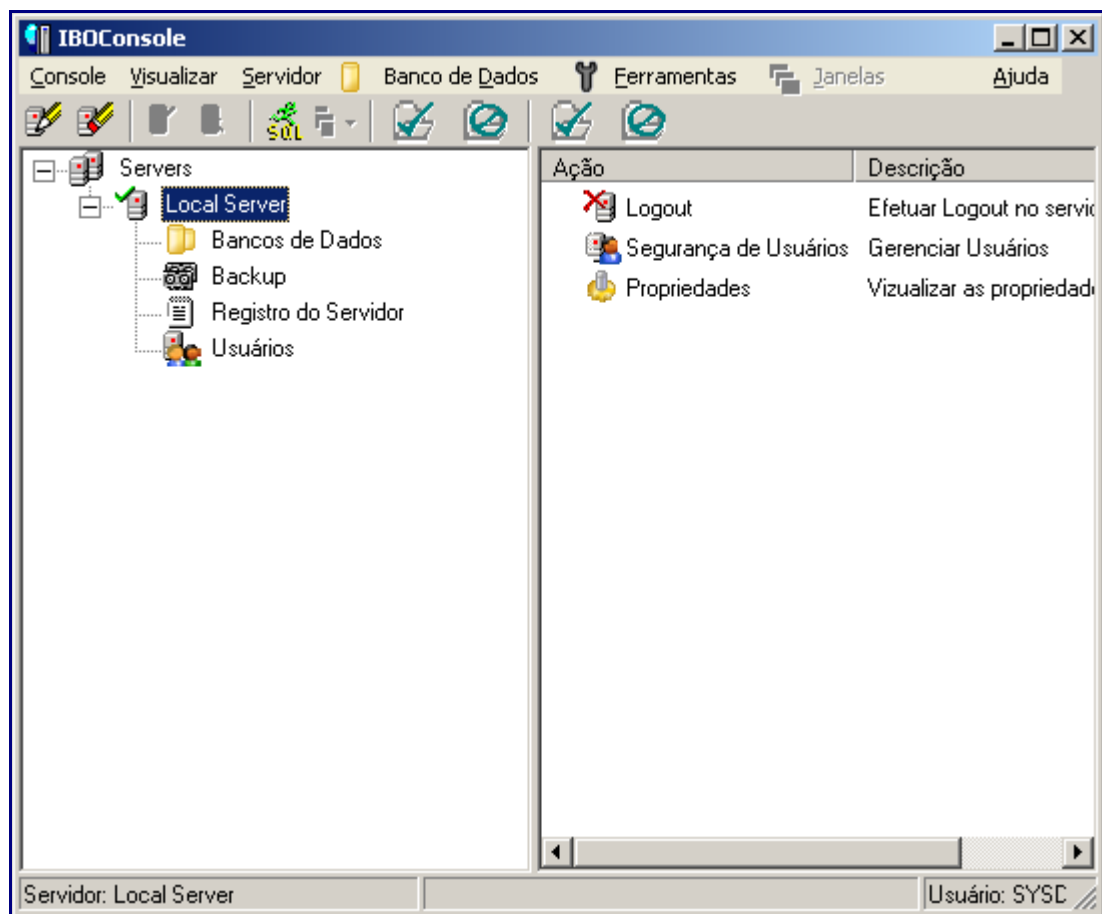
A tela é a mesma, basta optar por **Remote Server** e preencher os campos:

- Nome do Servidor: Endereço IP ou FQDN (Nome qualificado de Domínio, tipo www.algo.com.br)
- Protocolo de Rede: normalmente TCP/IP
- Nome do Alias: Alias é um “apelido” para referenciar o servidor no IBOConsole
- Descrição: Opcional
- Usuário e Senha: conforme seu acesso ao servidor

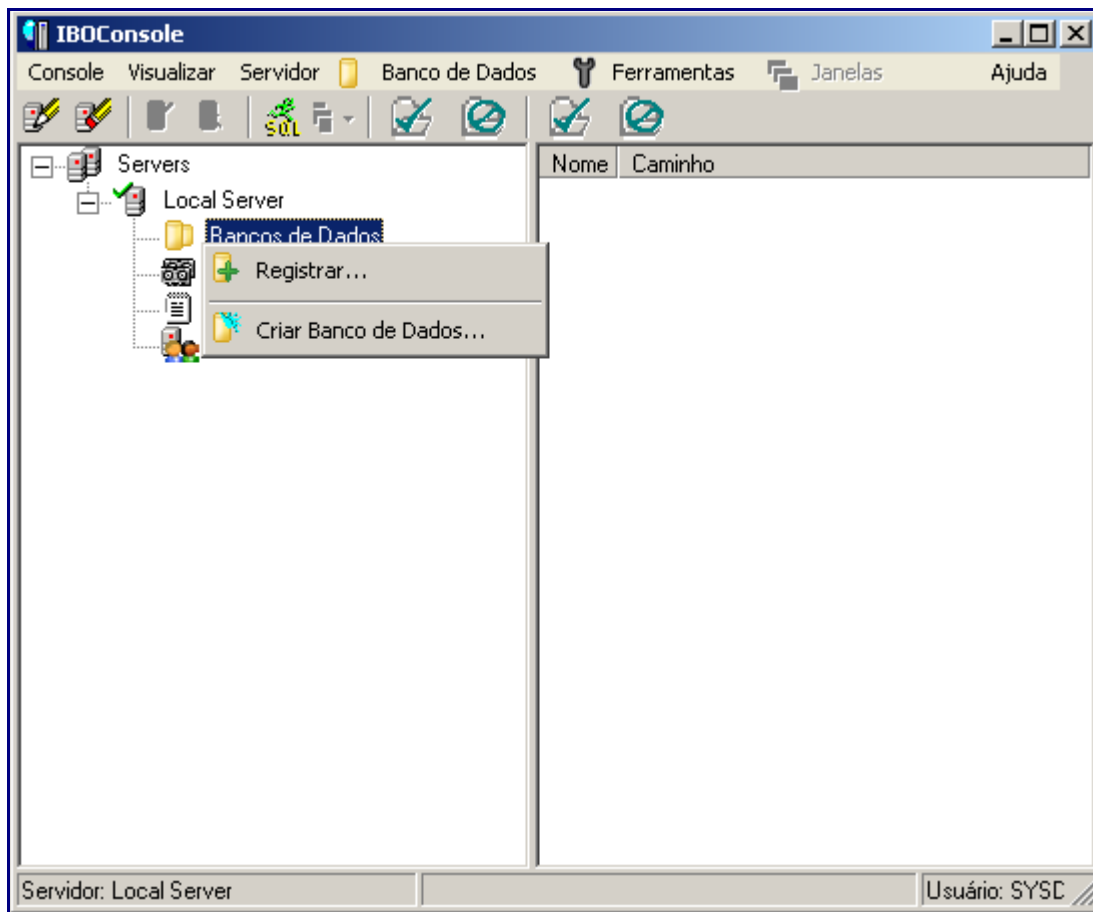


Criando/Registrando um Banco de Dados no Servidor

Após o registro, o IBOConsole apresenta a seguinte tela:



Se no servidor existe não existe nenhum banco de dados, devemos **criar** o que utilizaremos. Entretanto, podemos querer utilizar um banco já existente, ou seja, vamos **registrar** um banco. Utilize o menu **Banco de Dados → Registrar**.ou **Banco de Dados → Criar Banco de Dados**.ou clicando como botão direito em **Banco de Dados** na janela à direita, dentro do servidor desejado.



Criando um Banco de Dados no Servidor

Optando por **Criar Banco de Dados...**, a tela a seguir surge:

Server: Local Server

File(s):

Filename(s)	Size (Pages)
C:\banco\deva.fdb	

Options:

Page Size	4096
Default Character Set	None
SQL Dialect	3

☒ Register database

Alias: Deva

OK Cancel

Pode parecer estranho o FIREBIRD solicitar vários “FileNames” (Nomes de arquivos), mas o FIREBIRD pode dividir o banco em vários arquivos para otimizar o desempenho (especialmente em bancos muito grandes). Não é o nosso caso. Os arquivos podem estar localizados em qualquer local, com qualquer nome. Entretanto, é um padrão utilizar a extensão **fdb** para arquivos do FIREBIRD. O Interbase utiliza a extensão **gdb**, então você pode se habituar a encontrar as duas formas.

No exemplo, utilizou-se:

- Filenames: C:\banco\deva.fdb → Diretório e arquivo onde será armazenado
- Alias: Deva → Apelido para nosso banco

Observação: A criação do banco também pode ser realizada através de comandos SQL, o que será visto mais adiante.

Registrando um Banco de Dados no Servidor

Optando por **Registrar...**, a tela a seguir surge:

Register Database and Connect

Server: Local Server

Database

File: C:\banco\DEVA.FDB

Alias Name: Deva

☒ Save Alias Information

Login Information

User Name: SYSDBA

Password: xxxxxxxxxxxx

Role: ☐ Case sensitive role name

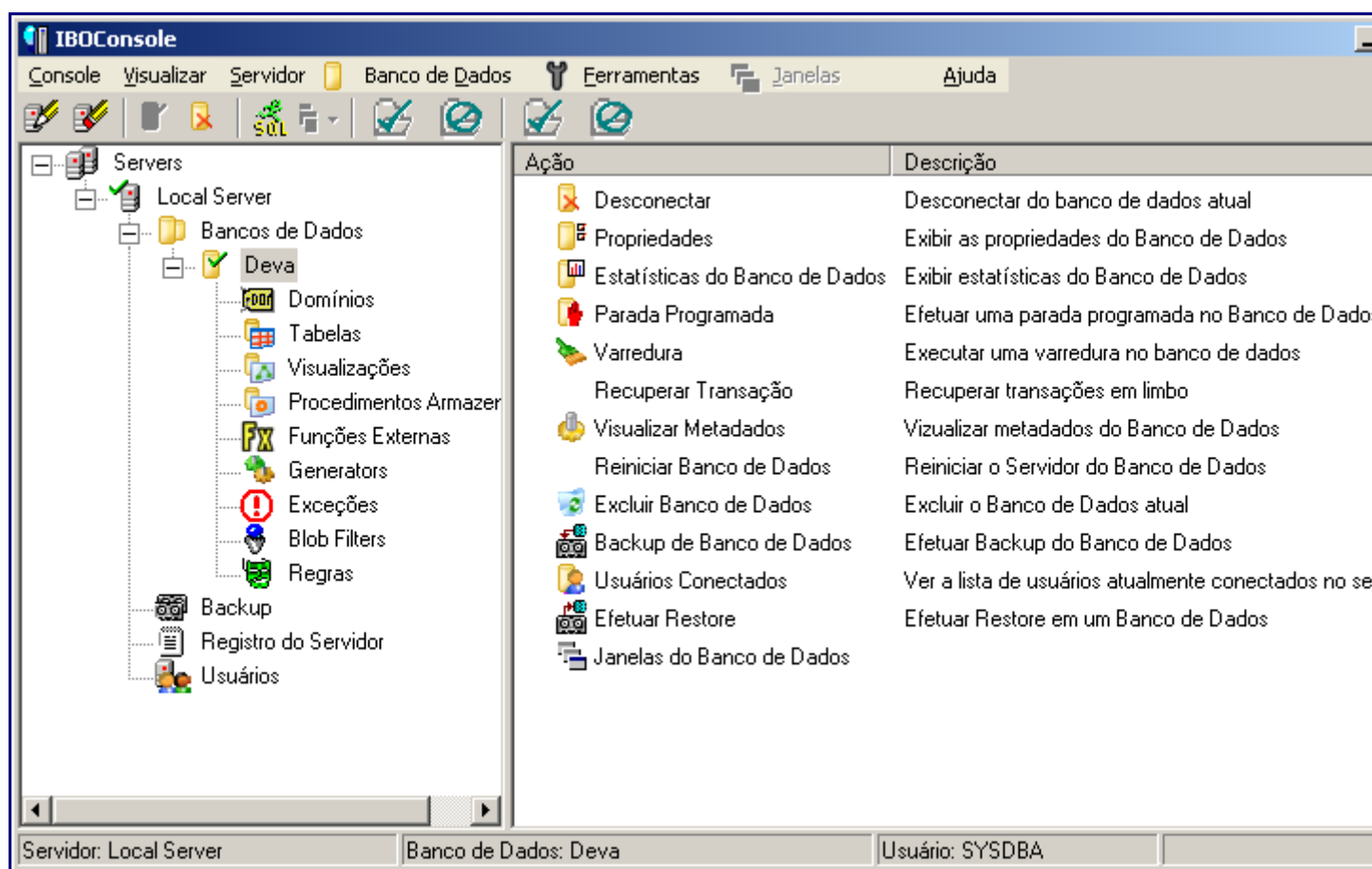
Default Character Set: UTF-8

OK Cancel

- File: C:\banco\deva.fdb
- Alias: Deva
- UserName: [Usuário para acesso]
- Password: [Senha do usuário]

Tela de Acesso ao Banco

Após a criação ou registro do Banco, todos os objetos e ações estarão disponíveis.

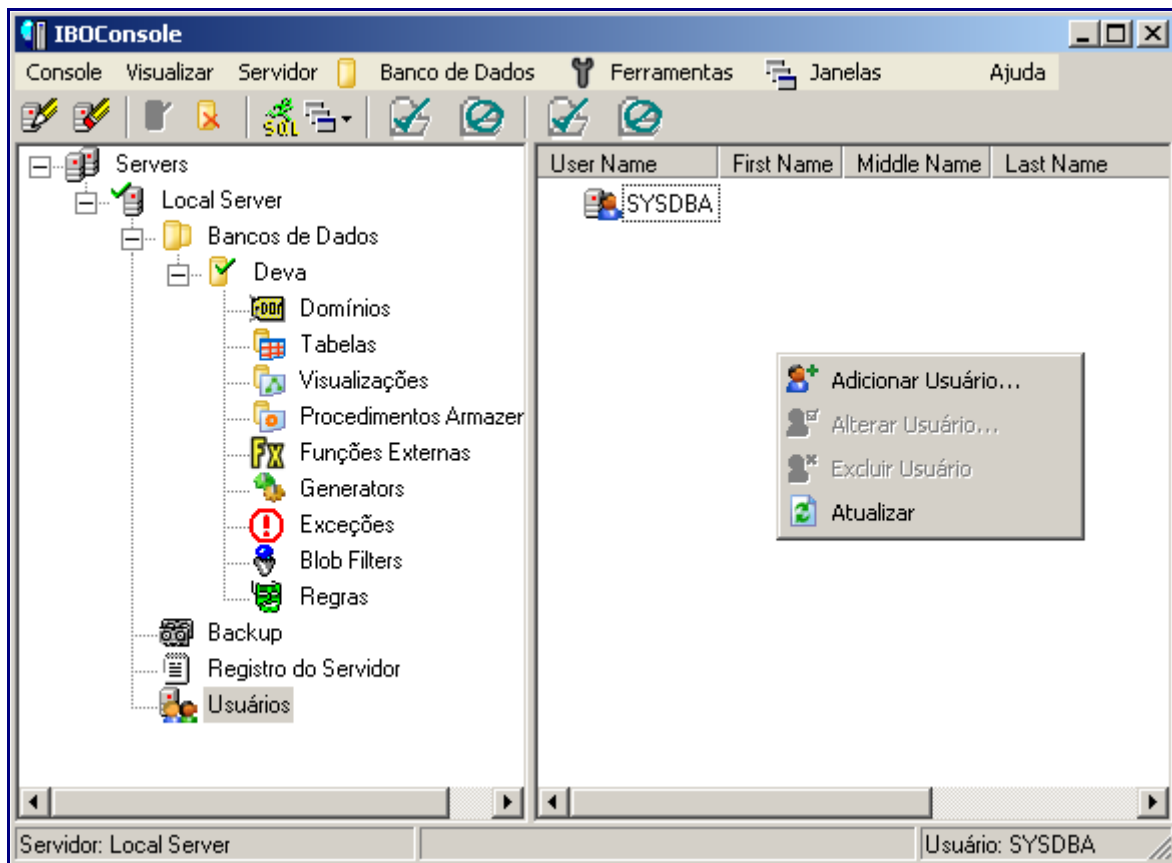


Selecionando o banco na janela da esquerda temos várias opções de consultas de operações de manutenção na janela da direita. Não é nosso foco nos determos nestas operações de manutenção.

Manutenção de Usuários

O IBOConsole permite sua criação de usuários e a definição de permissões a nível de objeto. Também é possível realizar estas operações através de comandos SQL.

Clicando em **Usuários** na lista à esquerda (os usuários são manipulados para cada servidor) teremos à direita a lista dos usuários deste servidor. Clicando com o botão direito na janela vazia podemos criar um novo usuário, ou clicando em cima de um existente podemos alterá-lo ou excluí-lo



A tela de criação é auto-explicativa; apenas os campos **Primeiro Nome**, **Nome do meio** e **Sobrenome** são opcionais

A tela de alteração é igual a de criação, sendo utilizada principalmente para alterar a senha.

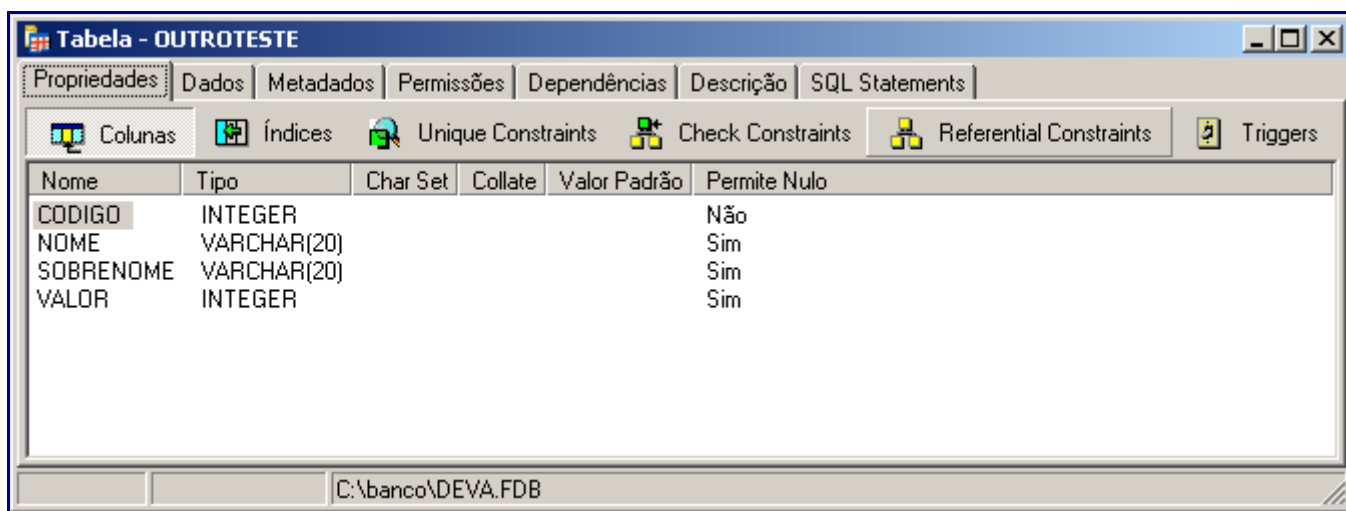
Manutenção de Tabelas

Clicando em **Tabelas** de um banco, na janela à esquerda, teremos a lista de tabelas do banco. Clicando com o botão direito sobre a tabela podemos ver suas propriedades.

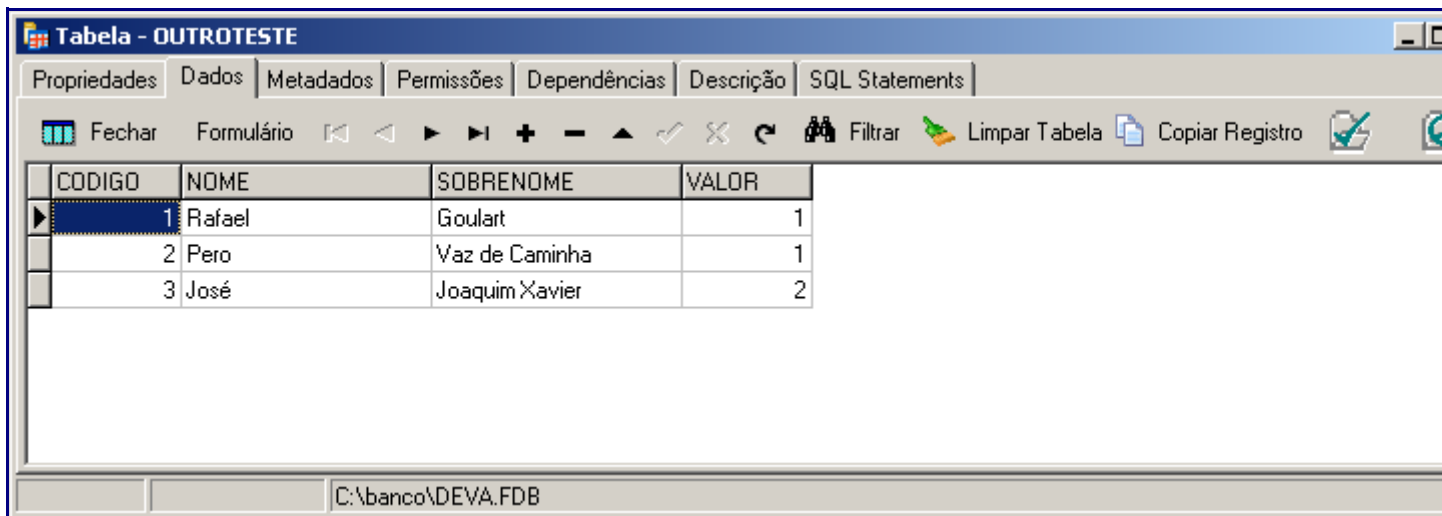


A primeira tela mostra a definição de campos da tabela. Clicando com o botão direito sobre eles é possível incluir novos campos, alterá-los ou excluí-los.

Outros detalhes sobre a tabela, como índices, constraints, etc, podem ser consultados.



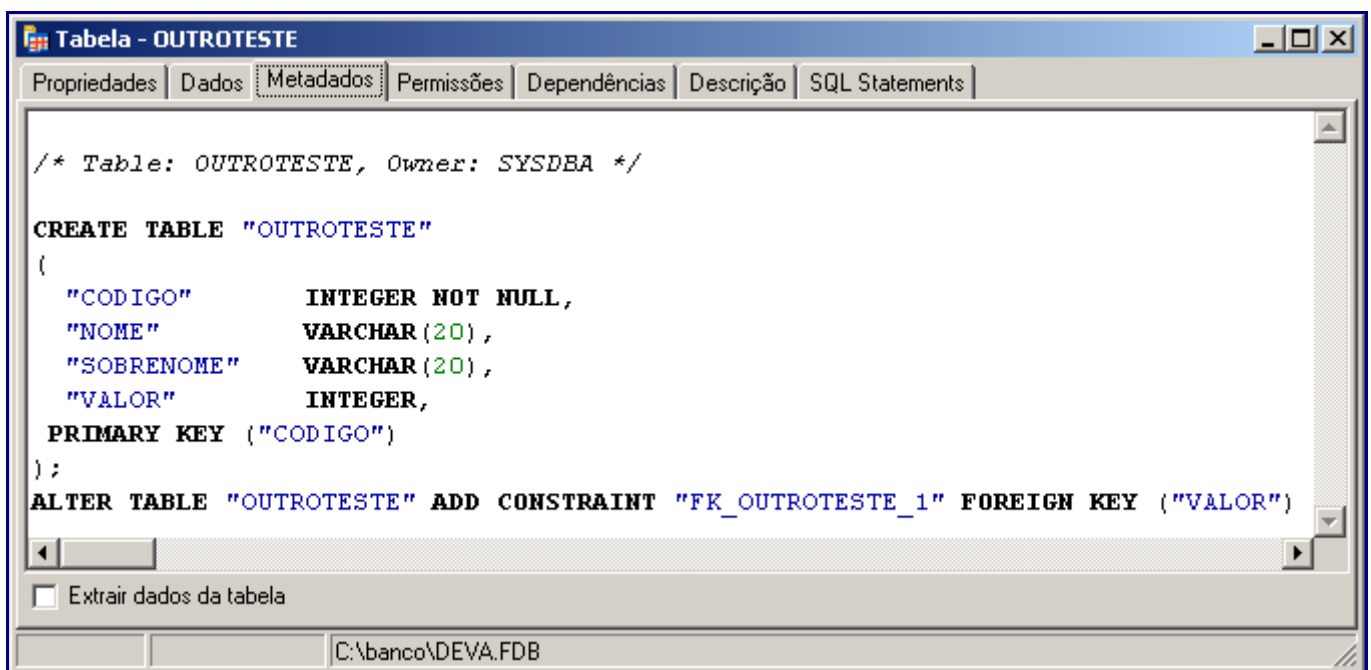
A guia Dados permite alterar os registros da tabela. Vários recursos estão disponíveis, e quem utiliza o Access vai perceber certa semelhança na edição.



	CODIGO	NOME	SOBRENOME	VALOR
▶	1	Rafael	Goulart	1
	2	Pero	Vaz de Caminha	1
	3	José	Joaquim Xavier	2

At the bottom of the window, the file path is shown: C:\banco\DEVA.FDB

A guia metadados mostra os comandos de criação da tabela. Bastante útil para recriar a tabela. A opção “Extrair dados da Tabela” agrega um comando INSERT para cada registro da tabela, ou seja, faz um “dump” dos dados.



```

/* Table: OUTROTESTE, Owner: SYSDBA */

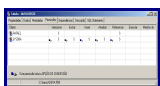
CREATE TABLE "OUTROTESTE"
(
  "CODIGO"          INTEGER NOT NULL,
  "NOME"            VARCHAR(20),
  "SOBRENOME"       VARCHAR(20),
  "VALOR"           INTEGER,
  PRIMARY KEY ("CODIGO")
);
ALTER TABLE "OUTROTESTE" ADD CONSTRAINT "FK_OUTROTESTE_1" FOREIGN KEY ("VALOR")

```

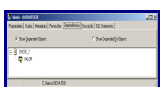
☐ Extrair dados da tabela

At the bottom of the window, the file path is shown: C:\banco\DEVA.FDB

A guia permissões exibe as permissões dos usuários para a tabela. Clicando com o botão direito sobre o usuário é possível alterar as permissões.

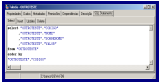


A guia Dependência exibe as relações de dependência da tabela com outras - tantos as que dependem dele quanto das quais ele depende.



A guia Descrição exibe um comentário opcional que pode ser anexado à tabela (útil para fins de documentação).

Por fim a última guia, SQL Stataments, dá a sintaxe de SELECT, INSERT, UPDATE e DELETE para a tabela. Muito prático para tabelas extensas.



Executando comandos SQL

O IBOConsole possui uma tela para execução de comandos SQL. É possível acessá-la de duas formas:

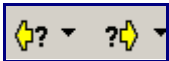
Menu Ferramentas → SQL Interativo



Botão na barra de ferramentas

Para executar comandos em determinado banco, é necessário que ele esteja selecionado antes de acessar o SQL Interativo.

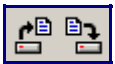
Vejamos os botões mais importantes da barra de ferramentas no **SQL Interativo**:



Navegar entre as sentenças executados



Executar sentença SQL



Salvar ou abrir scripts SQL



Commit - confirmar execução das sentenças

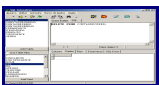


Rollback - cancelar execução das sentenças

A tela é dividida em quatro áreas:

- Lista de tabelas
- Lista de campos da tabela selecionada
- Região de Consultas (Guia SQL - padrão - e Query Builder)
- Região de retorno (dados, informações e erros das consultas)

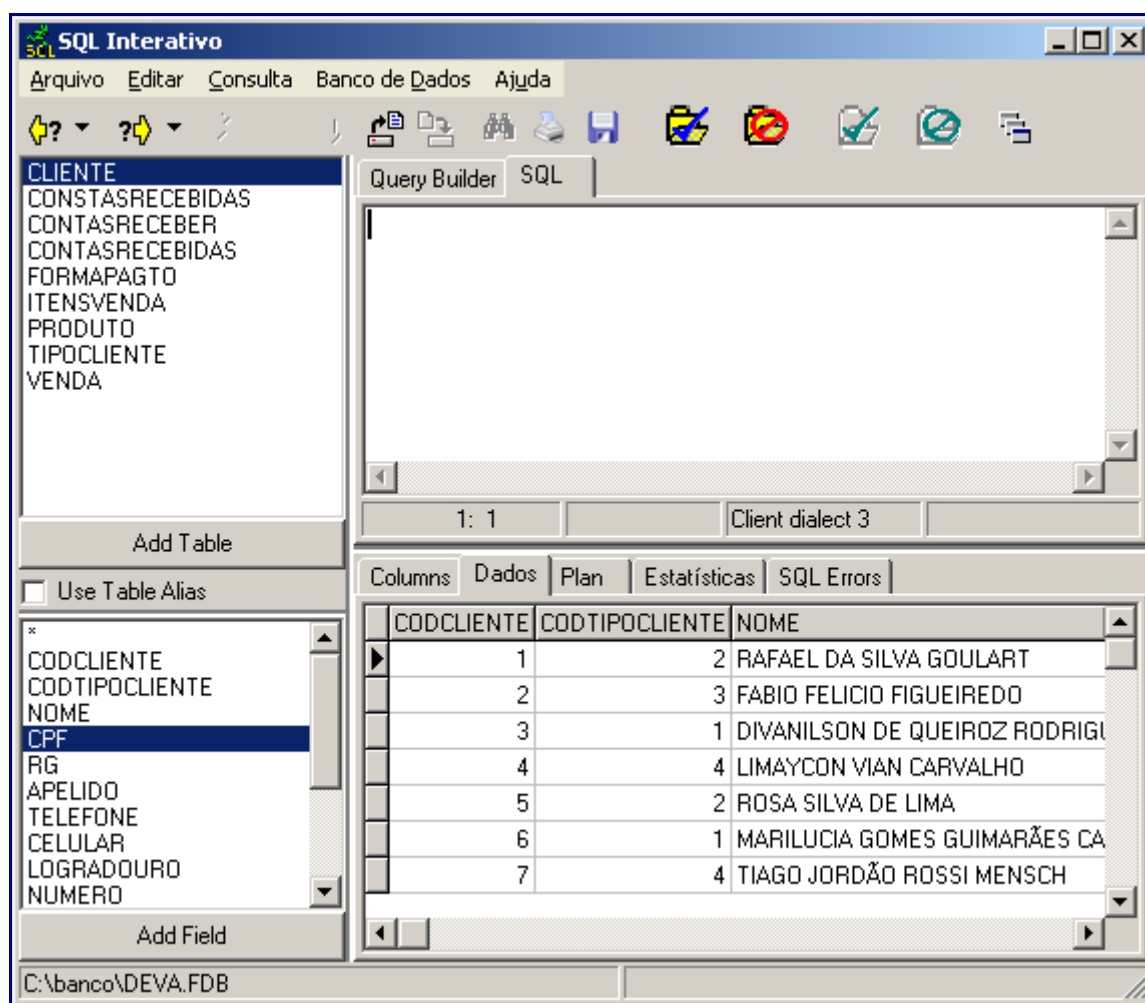
A sentença SQL deve ser digitada na guia correspondente, e após se executa o botão **Executar sentença SQL**.



Quando uma sentença contiver erros de sintaxe ou de integridade referencial, as mensagens serão retornadas na tela abaixo:

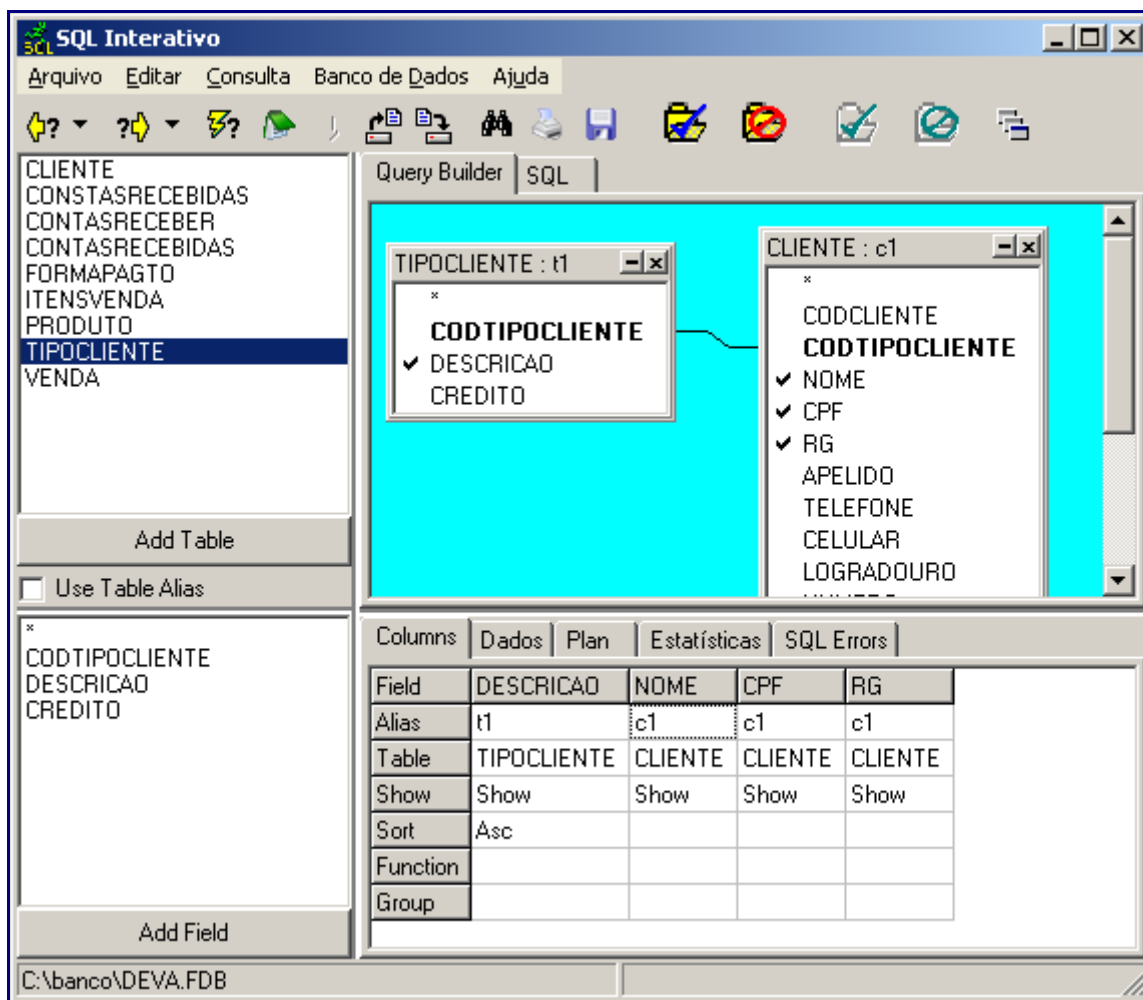


Já quando houver resultado positivo para um SELECT, os dados também serão exibidos abaixo:



Uma funcionalidade que alguns gostam é o **Query Builder (Construtor de Consultas)**, uma interface visual para construção de consultas. Ela é **Drag And Drop (Segure e Arraste)**: arraste as tabelas para a área desenho, arraste os campos para tela de baixo para selecioná-los arraste os campos de uma tabela para outra para relacioná-las. Use o botão direito para conhecer outras funcionalidades desta ferramenta.

Após executar a consulta, clique na guia **SQL** para ver a sentença em formato SQL.



Importante: ao executar consultas que alteram dados (INSERT, UPDATE, DELETE) estas só serão efetivadas após o uso do botão **Commit**.

SQL Básico

Todos os comandos a seguir devem ser executados a partir de um cliente. O Firebird possui um cliente em modo texto, o ISQL, que pode ser acessado na instalação detalhada neste curso a partir do **Menu Iniciar → Programas → Firebird_1_5 → Firebird ISQL Tool** diretamente através do executável em **C:\Arquivos de programas\Firebird\Firebird_1_5\bin\isql.exe**.

Evidentemente, é mais prático utilizar uma interface gráfica para executar os comandos. Em nosso curso, utilizaremos o IBOConsole.

Uma referência à todos os comando da linguagem é encontrada em [Apostila Firebird 1.0](#). As sintaxes apresentadas neste curso são simplificadas e objetivas, omitindo recursos avançados não utilizados nos exemplos.

Alguns padrões de sintaxe

Os comando SQL precisam terminar em **ponto-e-vírgula (;)**. Em algumas situações como quando executados isoladamente ou na tela de comandos SQL do IBOConsole é possível ignorar o ponto-e-vírgula. Porém, em scripts ou no ISQL isto é imprescindível. Para a criação do hábito, todos os

comandos aqui apresentados terão a terminação padrão.

Strings (textos) devem estar envolvidos em **aspas simples/apóstrofo (')**. Caso seja necessário colocar aspas simples/apóstrofo no texto, utiliza-se duas vezes. Exemplo:

```
'Isto é um texto válido'  
'Caixa d''água'  
'Ops... Caixa d'água com erro de sintaxe'
```

Comentários (texto que não será executado) no SQL é criado colocando entre **/*** e ***/**.

```
/* Isto é um comentário */
```

SQL DDL Básico (Manipulando Objetos do BD)

Criando um Banco de Dados (CREATE DATABASE)

Utilizando o IBOConsole esta operação já foi explicada em seu pequeno tutorial, que você pode [acessar aqui](#). Para criar “na mão grande”, ou para utilização em um script de criação de banco, utilizamos a sintaxe a seguir do comando **CREATE DATABASE** (a sintaxe é simplificada):

```
CREATE DATABASE "CAMINHO\NOME_DO_BANCO.FDB";
```

Por exemplo, criando o banco de dados do DEVA no diretório C:\BANCO:

```
CREATE DATABASE "C:\BANCO\DEVA.FDB";
```

Se quiser utilizar o banco através do IBOConsole, não deixe de registrá-lo ([lembre aqui como fazê-lo](#)).

Conectando a um Banco de Dados (CONNECT)

Ao clicar duas vezes num Banco de Dados no IBOConsole, internamente estaríamos conectando ao banco de dados com o comando **CONNECT**:

```
CONNECT "CAMINHO\NOME_DO_BANCO.FDB" USER "USUARIO" PASSWORD "SENHA";
```

Por exemplo, criando o banco de dados do DEVA no diretório C:\BANCO:

```
CONNECT "C:\BANCO\DEVA.FDB" USER "SYSDBA" PASSWORD "masterkey";
```

Importante: o usuário não é case sensitive, mas a senha é. Em bom português, não faz diferença se você digita o usuário em maiúsculas ou minúsculas, mas a senha precisa ser digitada como foi criada.

- Usuário **SYSDBA** ou **sysdba** não faz diferença, mas;
- Senha **masterkey** ou **MASTERKEY** faz muita diferença

O comando **CONNECT** tem mais sentido num script ou utilizando o ISQL em modo texto, mas importante conhecê-lo exatamente para editar scripts que executam comandos em lote.

Criando Tabelas (CREATE TABLE)

Antes de criar tabelas, é preciso conhecer os **Tipos de Campos** (ou **Domínios**) disponíveis no

FIREBIRD.

Observação: uma explicação detalhada sobre este assunto é encontrado na [Apostila Firebird 1.0](#).

Tipos de Dados (Domínios Padrão) do FIREBIRD

- **CHAR(n)** - String (texto) de tamanho fixo “n”. Limite 32767,32k. Usado quando o texto armazenado é sempre do mesmo tamanho, como em UF, CPF, CEP
- **VARCHAR(n)** - String (texto) de tamanho variável até o máximo de “n”. Limite 32767,32k. Usado quando não se sabe o tamanho exato do texto a ser armazenado, mas apenas seu tamanho máximo, como em Nome, Endereço, etc.
- **DATE** - Data (o formato armazenado é aaaa-mm-dd, ou 2000-12-31 para 31 de dezembro de 2000)
- **TIME** - Hora
- **TIMESTAMP** - Data e Hora Simultaneamente
- **DECIMAL(n,d)** ou **NUMERIC(n,d)** - números com precisão decimal (n - antes da vírgula, d - casas decimais)
- **SMALLINT** - Dados numéricos inteiros pequenos, na faixa de -32768 a 32767.
- **INTEGER** - Dados numéricos inteiros grandes (32bits), na faixa de -2.147.483.648 até

2.147.483.648

- **FLOAT** - Dados numéricos com precisão simples de 7 dígitos (casas decimais).
- **DOUBLE PRECISION** - Dados numéricos que exigem grande precisão (casas decimais). 64 bits.
- **BLOB** - Este tipo de campo é o tipo indicado para armazenar Textos Grandes “Memos”, Fotos, Gráficos, Ícones, isto é, aparentemente não tem um tipo de dado que não possa ser armazenado no Campo Blob. Campos Blob’s não podem ser indexados. Limite: 64k.
Subtipos:
 - SUB_TYPE 0 - Formatos binários: fotos, sons, etc;
 - SUB_TYPE 1 - Textos

Sintaxe CREATE TABLE (Criar Tabela)

A criação de uma tabela simples segue a sintaxe abaixo:

```
CREATE TABLE NOMETABELA (  
NOMECAMPO1 TIPO,  
NOMECAMPO2 TIPO,  
NOMECAMPO3 TIPO,  
NOMECAMPO4 TIPO  
);
```

Perceba que a lista dos campos está envolvida em parenteses, e que após cada campo há uma **vírgula (,)**, e após o último campo não há. Um exemplo:

```
CREATE TABLE CLIENTE (  
CODIGO INTEGER,  
NOME VARCHAR(40),  
TIPO INTEGER,  
ENDERECO VARCHAR(70),  
CIDADE VARCHAR(40),  
UF CHAR(2),  
OBSERVACAO BLOB SUB_TYPE 1,  
DATANASCIMENTO DATE,  
DATACADASTRO DATE
```

```
);
```

Opções comuns para campos na criação de tabelas

Além da definição do tipo, algumas opções são comumente utilizadas na criação de campos nas tabelas. Vejamos os mais comuns:

- **NOT NULL:** *NÃO NULO* Utilizado para forçar que o campo seja sempre preenchido com um valor, não sendo permitidos valores nulos. **SEMPRE** necessário quando o campo é **chave primária**.
- **DEFAULT:** *Valor Padrão* Utilizado para informar um valor padrão para o campo. Pode ser utilizada uma função, com será visto mais tarde, para determinar o valor

```
CREATE TABLE CLIENTE (  
  CODIGO INTEGER NOT NULL,  
  TIPO INTEGER NOT NULL,  
  NOME VARCHAR(40) NOT NULL,  
  ENDERECO VARCHAR(70),  
  CIDADE VARCHAR(40),  
  UF CHAR(2) DEFAULT 'BA',  
  OBSERVACAO BLOB SUB_TYPE 1,  
  DATANASCIMENTO DATE,  
  DATACADASTRO DATE  
);
```

Criando chaves primárias (PK - Primary Key)

A criação de chaves primárias é feita logo após a lista de campos. A sintaxe é:

```
PRIMARY KEY (campo)  
PRIMARY KEY (campo1,campo2) -> para chaves compostas  
);
```

Exemplo:

```
CREATE TABLE CLIENTE (  
  CODIGO INTEGER NOT NULL,  
  NOME VARCHAR(40) NOT NULL,  
  TIPO INTEGER NOT NULL,  
  ENDERECO VARCHAR(70),  
  CIDADE VARCHAR(40),  
  UF CHAR(2) DEFAULT 'BA',  
  OBSERVACAO BLOB SUB_TYPE 1,  
  DATANASCIMENTO DATE,  
  DATACADASTRO DATE,  
  PRIMARY KEY (CODIGO)  
);
```

Criando chaves estrangeiras (FK - Foreign Key)

A criação de chaves primárias é feita logo também após a lista de campos. A sintaxe é:

```
/* Sintaxe básica*/  
FOREIGN KEY (campo) REFERENCES tabela_estrangeira(campo_tabela_estrangeira)  
/* Sintaxe Extendida */  
FOREIGN KEY (campo) REFERENCES tabela_estrangeira(campo_tabela_estrangeira) ON  
UPDATE {action} ON DELETE {action}  
);
```

Na sintaxe extendida, define-se o que acontecerá com o registro desta tabela se acontecer a

atualização da chave estrangeira (ON UPDATE) ou na exclusão do registro correspondente na tabela estrangeira (ON DELETE). Vejamos os resultados:

- **NO ACTION** (Sem ação) - Não faz nada, apenas impede a ação (restrição de alteração ou exclusão da chave na tabela estrangeira);
- **CASCADE** (cascatear) - No caso de atualização da chave na tabela estrangeira, atualiza nesta tabela; no caso de exclusão do registro na tabela estrangeira, *EXCLUI TODOS OS REGISTROS RELACIONADOS NESTA TABELA* (note que pode ser útil ou perigoso...);
- **SET DEFAULT** (define padrão) - Define para um valor padrão aqui informado em caso de alteração ou exclusão da chave na tabela estrangeira;
- **SET NULL** (define para nulo) - Define para nulo em caso de alteração ou exclusão da chave na tabela estrangeira.

Só é possível criar uma chave estrangeira se já houver sido criada a tabela estrangeira.

```
CREATE TABLE TIPOCLIENTE (  
  CODIGOTIPO INTEGER NOT NULL,  
  DESCRICAO VARCHAR(20),  
  PRIMARY KEY (CODIGOTIPO)  
);  
  
CREATE TABLE CLIENTE (  
  CODIGOCLIENTE INTEGER NOT NULL,  
  NOME VARCHAR(40) NOT NULL,  
  TIPO INTEGER NOT NULL,  
  ENDERECO VARCHAR(70),  
  CIDADE VARCHAR(40),  
  UF CHAR(2) DEFAULT 'BA',  
  OBSERVACAO BLOB SUB_TYPE 1,  
  DATANASCIMENTO DATE,  
  DATACADASTRO DATE,  
  PRIMARY KEY (CODIGOCLIENTE),  
  FOREIGN KEY (TIPO) REFERENCES TIPOCLIENTE (CODIGOTIPO)  
);
```

Chaves Primárias Autonumeradas

Alguns bancos como o MS Access ou o MySQL possuem parâmetros simples para criar campos autonumerados, muito úteis para chaves primárias. Mas no FIREBIRD o trabalho é maior.

Um dos objetos existente no FIREBIRD é o **GENERATOR** (algo como “Gerador”), que nada mais é que um acumulador de um número inteiro. Uma técnica simples para criar um campo autonumerado é utilizar o objeto **TRIGGER** (Gatilho), que executa uma ação vinculada a um evento na tabela.

O truque é o seguinte:

```
CREATE GENERATOR {TABELA}_GEN;  
SET TERM ^ ;  
CREATE TRIGGER "TRIG_{TABELA}_GEN" FOR "CLIENTE"  
ACTIVE BEFORE INSERT POSITION 0  
AS  
begin  
  IF (new.{CAMPOCHAVE} IS NULL) then  
    begin  
      new.{CAMPOCHAVE} = gen_id( {TABELA}_GEN, 1 );  
    end  
  end  
end  
^  
COMMIT WORK ^  
SET TERM ;^
```

Sendo:

- {TABELA} = Tabela a criar a autonumeração
- {CAMPOCHAVE} = Chave Primária da Tabela

Exemplo para tabela cliente que estamos criando:

```
CREATE GENERATOR CLIENTE_GEN;  
SET TERM ^ ;  
CREATE TRIGGER "TRIG_CLIENTE_ID" FOR "CLIENTE"  
ACTIVE BEFORE INSERT POSITION 0  
AS  
begin  
    IF (new.CODIGOCLIENTE IS NULL) then  
        begin  
            new.CODIGOCLIENTE = gen_id( CLIENTE_GEN, 1 );  
        end  
    end  
end  
^  
COMMIT WORK ^  
SET TERM ;^
```

Alterando TABELAS (ALTER TABLE)

O comando ALTER TABLE é utilizado e conjunto com a sintaxe de criação:

```
/* Incluindo um campo */  
ALTER TABLE NOMETABELA ADD NOVOCAMPO5 TIPO;  
  
/* Excluindo um campo */  
ALTER TABLE NOMETABELA DROP NOVOCAMPO5;  
  
/* Incluindo e excluindo ao mesmo tempo  
ALTER TABLE NOMETABELA DROP NOVOCAMPO5, */  
    ADD NOVOCAMPO6 TIPO;  
  
/* Alterando o nome de um campo */  
ALTER TABLE NOMETABELA ALTER CAMPO5 TO CAMPO6;  
  
/* Adicionando uma chave primária */  
ALTER TABLE NOMETABELA ALTER PRIMARY KEY (CAMPO1);  
  
/* Adicionando uma chave estrangeira */  
ALTER TABLE NOMETABELA ALTER FOREIGN KEY (CAMPO1) REFERENCES TABELAESTRANGEIRA  
(CAMPOCHAVE);
```

Alguns exemplos:

```
/* Incluindo um campo */  
ALTER TABLE CLIENTE ADD EMAIL VARCHAR(30) NOT NULL;  
  
/* Excluindo um campo */  
ALTER TABLE CLIENTE DROP LOGRADOURO;  
  
/* Incluindo e excluindo ao mesmo tempo */  
ALTER TABLE CLIENTE ADD EMAIL VARCHAR(30) NOT NULL, DROP LOGRADOURO;  
  
/* Alterando o nome de um campo */  
ALTER TABLE CLIENTE ALTER LOGRADOURO TO ENDERECO;  
  
/* Adicionando uma chave primária */  
ALTER TABLE CLIENTE ALTER PRIMARY KEY (CODIGO);  
  
/* Adicionando uma chave estrangeira */
```

```
ALTER TABLE CLIENTE ALTER FOREIGN KEY (CODTIPOCLIENTE) REFERENCES TIPOCLIENTE (CODTIPOCLIENTE);
```

Excluindo TABELAS, BANCOS e outros objetos

O comando de exclusão é o **DROP**. Ele pode ser executado para vários objetos:

```
/* Excluindo uma tabela */
DROP TABLE nome_da_tabela;

/* Excluindo um generator */
DROP GENERATOR nome_da_tabela;

/* Excluindo uma TRIGGER */
DROP TRIGGER nome_trigger

/* Excluindo um banco */
DROP DATABASE "CAMINHO\NOME_DO_BANCO.FDB";
```

Exclusão é algo radical, em especial de bancos inteiros. Deve ser realizada com cuidado. Após a confirmação da transação (**COMMIT**), não haverá volta, a não ser por backup.

SQL DML Básico (Manipulando Dados)

Inserindo Dados (INSERT)

Para inserir dados em uma tabela utilizamos o comando **INSERT**. A sintaxe do comando **INSERT** utiliza três blocos:

```
/* Definindo tabela */
INSERT INTO TABELA
/* Lista de campos separados por vírgula - se não for informada, serão todos os campos */
( CAMPO1, CAMPO2, CAMPO3, CAMPO4 )
/* Lista de valores a serem inseridos, separados por vírgula */
VALUES ('valor1', 2, '2006-12-31', NULL);
```

Textos devem estar entre **apóstrofes (')**. As datas no formato **aaaa-mm-dd**, também entre apóstrofes.

Valores NULL *Os campos não informados receberão o valor **nulo (NULL)**. Se o campo foi criado com a opção **NOT NULL**, isto acarretará um erro. Se possuir a opção **valor padrão (DEFAULT)**, ao invés de **NULL** armazenará o valor **DEFAULT** especificado. Ainda, se o campo possuir uma **TRIGGER** de inserção vinculada, poderá ter um valor automaticamente calculado. Nos exemplos que utilizamos de criação de tabela, utilizamos um **GENERATOR** em conjunto com uma **TRIGGER** que cria um campo “**autonumerado**“, sempre que o valor inserido seja **NULL**. Campos definidos como **PRIMARY KEY**, por serem obrigatoriamente **NOT NULL**, evidentemente terão o mesmo tratamento.*

Vejamos um exemplo considerando a tabela abaixo:

```
CREATE TABLE ALUNO (
MATRICULA CHAR(9) NOT NULL,
NOME VARCHAR(50),
```

```
DATANASC DATE,
DISCIPLINAS INTEGER
);
```

O código para inserir um registro pode ser algum dos abaixo, conforme a limitação de não inserir valores nulos para o campo MATRICULA:

```
/* Todos os campos, sem especificar */
INSERT INTO ALUNO
VALUES ('200304521','MARIA DA SILVA','1978-09-05',3);

/* Todos os campos, especificados */
INSERT INTO ALUNO (MATRICULA, NOME, DATANASC, DISCIPLINAS)
VALUES ('200412453','JOÃO DA SILVA','1980-10-04', 5);

/* Omitindo a data de nascimento */
INSERT INTO ALUNO (MATRICULA, NOME, DISCIPLINAS)
VALUES ('200103345','SHEILA DA SILVA', 2);

/* As inserções a seguir geram erros */

INSERT INTO ALUNO
VALUES ('200301248','FÁBIO DA SILVA','1976-08-18');
/*
Count of read-write columns does not equal count of values
(Contagem de colunas não é igual à contagem de valores)
Existem menos campos ou valores que o necessário
*/

INSERT INTO ALUNO
VALUES ('200501354','MAGDA DA SILVA','1976-13-18',4);
/*
conversion error from string "1976-13-18"
(erro de conversão da string "1976-13-18")
Data está errada, não pode ser convertida de texto em data (mês 13 não existe!)
*/

INSERT INTO ALUNO
VALUES ('200501354','JOAQUIM JOSÉ DA SILVA XAVIER FILHO SOBRINHO NETO PARENTE
DISTANTE','1976-12-18',4);
/*
arithmetic exception, numeric overflow, or string truncation
(excessão aritmética, sobrecarga numérica ou truncamento de string)
Algun valor extrapolou o limite de armazenamento - no caso, um nome com mais de
50 caracteres
*/
```

Quando se viola uma chave primária - informada em duplicidade - ou chave estrangeira - inserindo um valor que não é encontrado na tabela estrangeira - também se recebe mensagens de erro:

violation of PRIMARY or UNIQUE KEY constraint “INTEG_183” on table “TIPOCLIENTE”
(violação de restrição de CHAVE PRIMÁRIA ou ÚNICA “INTEG_183” na tabela “TIPOCLIENTE”)

violation of FOREIGN KEY constraint “INTEG_187” on table “CLIENTE”
(violação de restrição de CHAVE ESTRANGEIRA “INTEG_187” na tabela “CLIENTE”)

Selecionando Dados (SELECT)

Seleção é a operação mais utilizada em SQL. Veremos a seguir suas sintaxes mais básicas, criando gradativamente seleções mais complexas. O comando chave é o **SELECT**.

Sintaxe básica

```
/* Selecionar */
SELECT
/* Lista de campos separados por vírgula */
CAMPO1, CAMPO2, CAMPO3
/* Da tabela */
FROM TABELA;
```

Para selecionar todos os campos, utilize o **asterisco “*”**.

A consulta abaixo seleciona todos registros:

```
SELECT * FROM TABELA;
```

É possível também informar o nome dos campos de forma completa, especificando o “TABELA.CAMPO”. Isto pode parecer redundante, mas com a junção de várias tabelas é necessário evitar a ambigüidade, ou seja, fazer referência a campos que possuem o mesmo nome em duas tabelas de forma clara e específica.

```
SELECT TABELA.CAMPO1, TABELA.CAMPO2, TABELA.CAMPO3 FROM TABELA;
```

É possível, para facilitar, criar **alias (apelidos)** para tabelas e campos, facilitando sua referência. Tabelas ganham apelidos colocando-os logo em frente à tabela, e campos através da palavra chave **AS (COMO)**. Campos calculados, concatenados são freqüentemente referenciados por **alias**.

```
SELECT
A.MATRICULA, A.NOME AS NOMEALUNO
FROM ALUNO A;
```

```
SELECT
A.MATRICULA, A.NOME AS NOMEALUNO, C.CODCURSO
FROM ALUNO A, CURSO C
WHERE A.CODCURSO = C.CODCURSO;
```

Condições

A cláusula **WHERE** expressa a condição **ONDE**. É utilizada para:

- Filtrar os registros dada uma ou mais restrições;
- Relacionar tabelas.

Os campos que serão utilizados na filtragem não precisam ser exibidos no resultado da consulta.

A cláusula **WHERE** é usada **SEMPRE** após a lista de tabelas.

Filtrando dados

Para filtrar dados, é necessário usar operadores ou comparadores. Os operadores básicos são:

Operador	Descrição	Exemplo
=	igual a	CAMPO1 = 'RAFAEL' – CAMPO1 = CAMPO5 – CAMPO1 = 5
<>	diferente de	CAMPO1 <> 'RAFAEL' – CAMPO1 <> CAMPO5 – CAMPO1 <> 5
>	maior que	CAMPO1 > CAMPO5 – CAMPO1 > 5
>=	maior ou igual que	CAMPO1 >= CAMPO5 – CAMPO1 >= 5
<	menor que	CAMPO1 < CAMPO5 – CAMPO1 < 5

=<	menor ou igual que	CAMPO1 =< CAMPO5 – CAMPO1 =< 5
LIKE	como	Usado em comparação de trechos de texto → CAMPO1 LIKE '%RAFAEL%'
BETWEEN	entre	Entre dois valores → CAMPO1 BETWEEN 35 AND 50
IN	em	Contido numa lista de valores → CAMPO1 IN (1,2,3,4,5)

Observação: o símbolo percentual (%) é um coringa para pesquisas em texto, significa qualquer trecho de texto, inclusive nenhum.

Os campos utilizados no filtro podem, mas não precisam ser exibidos na seleção.

Exemplo:

```
/* Selecionar todos os campos */
SELECT *
/* da tabela ALUNO*/
FROM ALUNO
/* onde */
WHERE
/* o valor do campo DISCIPLINAS seja maior do que 3 */
DISCIPLINAS > 3;

/* Selecionar todos os campos */
SELECT *
/* da tabela ALUNO*/
FROM ALUNO
/* onde */
WHERE
/* o valor do campo NOME comece com 'RAFAEL' e continue com qualquer coisa
(inclusive nada) */
NOME LIKE 'RAFAEL%';
```

Para filtrar por mais de um campo, utilize as operadores lógicas:

Operador Descrição

OR	OU	Um filtro ou outro
AND	E	Um filtro e outro filtro

```
/* Selecionar todos os campos */
SELECT *
/* da tabela ALUNO*/
FROM ALUNO
/* onde */
WHERE
/* o valor do campo DISCIPLINAS seja maior do que 3 */
DISCIPLINAS > 3
/* E o valor do campo DATANASC seja maior do que 01/01/1980 */
AND DATANASC > '1980-01-01';
```

```
/* Selecionar todos os campos */
SELECT *
/* da tabela ALUNO*/
FROM ALUNO
/* onde */
WHERE
/* o valor do campo DISCIPLINAS seja maior do que 3 */
DISCIPLINAS > 3
```



```
/* E o valor do campo DATANASC seja maior do que 01/01/1980 */  
AND DATANASC > '1980-01-01';
```

```
/* Selecionar todos os campos */  
SELECT *  
/* da tabela ALUNO*/  
FROM ALUNO  
/* onde */  
WHERE  
/* o valor do campo MATRICULA seja igual a 200401542 */  
MATRICULA = '200401542'  
/* OU o valor do campo MATRICULA seja igual a 200306482*/  
OR MATRICULA = '200306482';
```

O uso de parênteses pode modificar a ordem de avaliação; os parênteses tem prioridade sobre o operador, assim como as expressões aritméticas.

Relacionando Tabelas

Relacionar tabelas é uma forma muito importante de cruzamento de dados, e também a mais simples. A forma de fazer isto é a seguinte:

1. Identifique a relação entre campo **CHAVE ESTRANGEIRA** e compare com o campo que é **CHAVE PRIMÁRIA** da **TABELA ESTRANGEIRA** (*);
2. Crie uma condição relacionando os campos das duas tabelas, com o operador “**igual**”;
3. Repita para cada relacionamento da tabela.

Este trabalho se torna mais minucioso quando existem várias tabelas relacionadas.

Considerando as tabelas:

```
CREATE TABLE CURSO (  
CODCURSO CHAR(3) NOT NULL,  
DESCRICAO VARCHAR(50),  
PRIMARY KEY (CODCURSO)  
);
```

```
CREATE TABLE TURMA (  
CODTURMA CHAR(3) NOT NULL,  
DESCRICAO VARCHAR(20),  
PRIMARY KEY (CODTURMA)  
);
```

```
CREATE TABLE ALUNO (  
MATRICULA CHAR(9) NOT NULL,  
NOME VARCHAR(50),  
DATANASC DATE,  
DISCIPLINAS INTEGER,  
CODCURSO CHAR(3),  
CODTURMA CHAR(3),  
PRIMARY KEY (MATRICULA),  
FOREIGN KEY (CODCURSO) REFERENCES CURSO (CODCURSO),  
FOREIGN KEY (CODTURMA) REFERENCES TURMA (CODTURMA)  
);
```

Uma consulta para verificar o aluno com seu curso e turma seria:

```
/* Selecionar campos MATRICULA, NOME, DESCRICAO do Curso e DESCRICAO Turma */  
SELECT ALUNO.MATRICULA, ALUNO.NOME, CURSO.DESCRICAO, TURMA.DESCRICAO  
/* das tabelas*/  
FROM ALUNO, CURSO, TURMA  
/* onde */
```

```
WHERE
/* vincula ALUNO a CURSO */
ALUNO.CODCURSO = CURSO.CODCURSO
/* E vincula ALUNO a TURMA */
AND ALUNO.CODTURMA = TURMA.CODTURMA;
```

Perceba que foi necessário especificar os campos DESCRICAO, CODTURMA, CODCURSO, pois aparecem mais de uma vez no resultado - existem em mais de uma tabela.

Ordenação (ORDER BY)

A cláusula **ORDER BY** permite ordenar o resultado por campos das tabelas envolvidas. Sua sintaxe é simples, e é **SEMPRE APÓS** o final das condições da cláusula **WHERE**, se ela existir - ou após onde ela estaria.

Pode-se informar mais de um campo; desta forma, será ordenado primeiramente pelo primeiro da lista, em seguida por cada um dos demais.

É possível determinar a forma de ordenação:

- **ASC**, ascendente, que é o padrão caso não seja informada, exibe os registros em ordem alfabética, numericamente crescente ou datas mais antigas primeiro, conforme o tipo do campo.;
- **DESC**, decendente, exibe os registros em ordem alfabética inversa, numericamente decrescente ou datas mais novas primeiro, conforme o tipo do campo.

```
SELECT ALUNO.MATRICULA, ALUNO.NOME, CURSO.DESCRICAO, TURMA.DESCRICAO
FROM ALUNO, CURSO, TURMA
WHERE
ALUNO.CODCURSO = CURSO.CODCURSO
AND ALUNO.CODTURMA = TURMA.CODTURMA
/* Ordenando pelo curso, depois pela turma, depois em ordem alfabética o nome dos alunos*/
ORDER BY CURSO.CODCURSO, TURMA.CODTURMA, ALUNO.NOME;
```

```
SELECT
*
FROM ALUNOS
/* Ordenando por alunos mais novos*/
ORDER BY DATANASC DESC;
```

Campos calculados, concatenação e funções

O FIREBIRD permite a execução de algumas operações entre

O Firebird possui um conjunto reduzido de funções

Agrupamento (GROUP e HAVING)

Atualizando Dados (UPDATE)

Para atualizar dados em uma tabela utilizamos o comando **UPDATE**. A palavra chave **SET (definir)** define os campos a serem alterados. A sintaxe do comando **UPDATE** é quase sempre utilizada em conjunto com a cláusula **WHERE**, restringindo os registros a serem atualizados. Isto nos gera 3 sintaxes básicas:

- Alterando **TODOS OS REGISTROS** - sem cláusula **WHERE**:

```

/* Definindo tabela */
UPDATE TABELA
/* Lista de campos a serem atualizados separados por vírgula */
SET
    CAMPO1 = 'Novo Valor',
    CAMPO2 = 45,
    CAMPO3 = '2006-12-31';

```

- Alterando **APENAS UM REGISTRO ESPECÍFICO** - cláusula **WHERE** filtrando a **chave primária** da tabela:

```

/* Definindo tabela */
UPDATE TABELA
/* Lista de campos a serem atualizados separados por vírgula */
SET
    CAMPO1 = 'Novo Valor',
    CAMPO2 = 45,
    CAMPO3 = '2006-12-31'
/* Filtrando a chave primária */
WHERE
    CAMPO_QUE_É_CHAVE_PRIMÁRIA = 4587

```

- Alterando **VÁRIOS REGISTROS** - cláusula **WHERE** filtrando um ou mais registros da tabela:

```

/* Definindo tabela */
UPDATE TABELA
/* Lista de campos a serem atualizados separados por vírgula */
SET
    CAMPO1 = 'Novo Valor',
    CAMPO2 = 45,
    CAMPO3 = '2006-12-31'
/* Filtrando vários registros */
WHERE
    CAMPO1 = 'Valor qualquer'
AND CAMPO2 > 2;

```

Um cuidado muito grande é necessário quando atualiza-se vários registros simultaneamente, para evitar que o **UPDATE** gere atualizações em mais registros que o desejado. Uma prática inteligente é realizar um **SELECT** com o mesmo filtro (cláusula **WHERE**) antes, testando se a atualização será realizada nos registros corretos. Exemplo:

```

/* Primeiro selecionamos os registros e conferimos o filtro */
SELECT * FROM TABELA
WHERE
    CAMPO1 = 'Valor qualquer'
AND CAMPO2 > 2;

/* Se os registros estiverem corretos, então realiza-se a atualização */
UPDATE TABELA
SET
    CAMPO1 = 'Novo Valor',
    CAMPO2 = 45,
    CAMPO3 = '2006-12-31'
WHERE
    CAMPO1 = 'Valor qualquer'
AND CAMPO2 > 2;

```

Outra forma é lembrar que as alterações só serão confirmadas após um **COMMIT**. Então, é possível selecionar os dados após a atualização, checá-los e apenas depois realizar o **COMMIT**.

Assim como o comando **INSERT**, o **UPDATE** gerará erros se o valor atualizado não respeitar o

tipo/tamanho do campo, chaves estrangeiras ou tiver erros de sintaxe.

Vejamos um exemplo considerando a tabela abaixo:

```
CREATE TABLE ALUNO (  
MATRICULA CHAR(9) NOT NULL,  
NOME VARCHAR(50),  
DATANASC DATE,  
DISCIPLINAS INTEGER,  
PRIMARY KEY (MATRICULA)  
);
```

Nos exemplos abaixo considera-se que existam alguns registros na tabela.

- Alterando **TODOS OS REGISTROS** - sem cláusula **WHERE**:

```
/* Definindo tabela */  
UPDATE ALUNO  
/* Lista de campos a serem atualizados separados por vírgula  
   Atualizando o campo DISPLINAS para 5  
*/  
SET  
    DISCIPLINAS = 5;
```

- Alterando **APENAS UM REGISTRO ESPECÍFICO** - cláusula **WHERE** filtrando a **chave primária** da tabela:

```
/* Definindo tabela */  
UPDATE ALUNO  
/* Lista de campos a serem atualizados separados por vírgula  
   Atualizando o campo DISPLINAS para 5  
*/  
SET  
    DISCIPLINAS = 5  
/* Filtrando apenas um registro através da chave primária MATRICULA */  
WHERE  
    MATRICULA = 217
```

- Alterando **VÁRIOS REGISTROS** - cláusula **WHERE** filtrando um ou mais registros da tabela:

```
/* Definindo tabela */  
UPDATE ALUNO  
/* Lista de campos a serem atualizados separados por vírgula  
   Atualizando o campo DISPLINAS para 5  
*/  
SET  
    DISCIPLINAS = 5  
/* Filtrando vários registros */  
WHERE  
    MATRICULA > 45  
AND DATANASC > '1980-01-01'
```

Excluindo Dados (DELETE)

Para atualizar dados em uma tabela utilizamos o comando **DELETE**. A sintaxe do comando **DELETE** é quase sempre utilizada em conjunto com a cláusula **WHERE**, restringindo os registros a serem atualizados. Funciona como um **SELECT** que apaga os registros selecionados. Isto nos gera 3 sintaxes básicas, como no **UPDATE**:

- Excluindo **TODOS OS REGISTROS** - sem cláusula **WHERE**:

```
/* Definindo tabela */  
DELETE FROM TABELA;
```

- Excluindo **APENAS UM REGISTRO ESPECÍFICO** - cláusula **WHERE** filtrando a **chave primária** da tabela:

```
/* Definindo tabela */  
DELETE FROM TABELA  
/* Filtrando a chave primária */  
WHERE  
    CAMPO_QUE_É_CHAVE_PRIMÁRIA = 4587
```

- Filtrando **VÁRIOS REGISTROS** - cláusula **WHERE** filtrando um ou mais registros da tabela:

```
/* Definindo tabela */  
DELETE FROM TABELA  
/* Filtrando vários registros */  
WHERE  
    CAMPO1 = 'Valor qualquer'  
AND CAMPO2 > 2;
```

Um cuidado muito grande é necessário quando exclui-se vários registros simultaneamente, para evitar que o **DELETE** exclua mais registros que o desejado. Uma prática inteligente é realizar um **SELECT** com o mesmo filtro (cláusula **WHERE**) antes, testando se a exclusão será realizada nos registros corretos (mesma dica usada para o **UPDATE**. Exemplo:

```
/* Primeiro selecionamos os registros e conferimos o filtro */  
SELECT * FROM TABELA  
WHERE  
    CAMPO1 = 'Valor qualquer'  
AND CAMPO2 > 2;  
  
/* Se os registros estiverem corretos, então realiza-se a atualização */  
DELETE FROM TABELA  
WHERE  
    CAMPO1 = 'Valor qualquer'  
AND CAMPO2 > 2;
```

Outra forma é lembrar que as alterações só serão confirmadas após um **COMMIT**. Então, é possível selecionar os dados após a atualização, checá-los e apenas depois realizar o **COMMIT**.

O comando **DELETE** gerará erros se o valor atualizado não respeitar o tipo/tamanho do campo, chaves estrangeiras ou tiver erros de sintaxe. Em especial, apagar registros de tabelas que são tabelas estrangeiras de outras podem gerar erros de **violação de chave estrangeira - (violation of FOREIGN KEY)**.

Vejamos um exemplo considerando a tabela abaixo:

```
CREATE TABLE ALUNO (  
    MATRICULA CHAR(9) NOT NULL,  
    NOME VARCHAR(50),  
    DATANASC DATE,  
    ISCIPLINAS INTEGER,  
    CODCURSO CHAR(3),  
    CODTURMA CHAR(3),  
    PRIMARY KEY (MATRICULA)  
);
```

Nos exemplos abaixo considera-se que existam alguns registros na tabela.

- Excluindo **TODOS OS REGISTROS** - sem cláusula **WHERE**:

```
/* Definindo tabela */  
DELETE FROM ALUNO;
```

- Excluindo **APENAS UM REGISTRO ESPECÍFICO** - cláusula **WHERE** filtrando a **chave primária** da tabela:

```
/* Definindo tabela */  
DELETE FROM ALUNO  
/* Filtrando apenas um registro através da chave primária MATRICULA */  
WHERE  
    MATRICULA = 217
```

- Excluindo **VÁRIOS REGISTROS** - cláusula **WHERE** filtrando um ou mais registros da tabela:

```
/* Definindo tabela */  
DELETE FROM ALUNO  
/* Filtrando vários registros */  
WHERE  
    MATRICULA > 45  
AND DATANASC > '1980-01-01'
```

SQL DCL Básico (Manipulando Usuários e Permissões)

Os comandos DCL têm a função de definir permissões de acesso a um usuário em tabelas de um banco de dados.

O comando GRANT é utilizado para permitir o acesso e o comando REVOKE para revogá-lo. Cada banco de dados possui algumas peculiaridades na sintaxe, mas os comando explanados a seguir são bastante genéricos.

Obs.: Para criação de usuários no FIREBIRD é necessário uma ferramenta específica. Veja em [manutencao de usuarios](#) como criar usuários com o IBOConsole.

Tipos de Privilégios

Permitindo acesso

A sintaxe básica de permissão é:

```
GRANT [lista_de_privilegios] ON [objeto(s)] TO [usuario];
```

Para permitir ao usuário FULANO realizar seleções na tabela CLIENTE, o comando seria:

```
GRANT SELECT ON CLIENTE TO FULANO;
```

Para permitir ao usuário FULANO realizar privilégios completos no banco DEVA, o comando seria:

```
GRANT ALL ON DEVA.* TO FULANO;
```

Revogando acesso

A sintaxe básica é:

```
REVOKE [lista_de_privilegios] ON [objeto(s)] TO [usuario];
```

Ou, como se pode perceber, a mesma sintaxe do GRANT, mas para fazer o contrário, revogar o privilégio.

Para cancelar a permissão do usuário FULANO de realizar seleções na tabela CLIENTE, o comando seria:

```
REVOKE SELECT ON DEVA.CLIENTE TO FULANO;
```

Para revogar o acesso completamente ao usuário FULANO banco DEVA, o comando seria:

```
REVOKE ALL ON DEVA.* TO FULANO;
```

Exemplo 01 - DEVA

DEVA (Devaldir Catarino) é o dono da cantina mais antiga de nossa faculdade, e alvo fácil de nossos estudos.

Abaixo temos um diagrama simples de um sistema para sua cantina.

Diagrama de Classes UML

Sistema de Gerenciamento da Cantina do DEVA

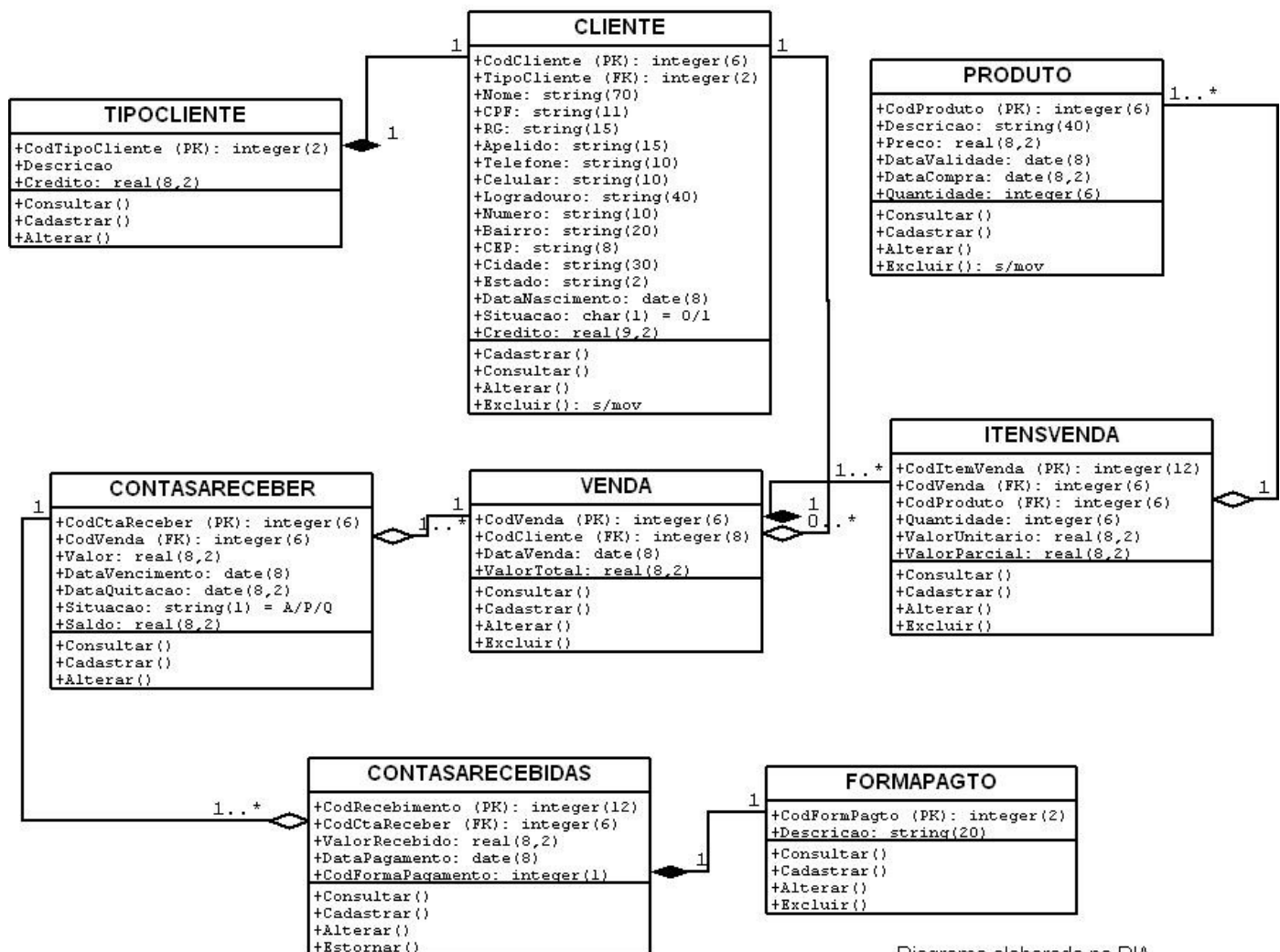


Diagrama elaborado no DIA

Este diagrama foi elaborado nas aulas de Análise de Projetos e Sistemas do Prof. Alexandre Monge, na [FASB](#).

O fonte deste diagrama (em formato do [DIA](#)) está disponível [aqui](#)

Criação das Tabelas do Banco de Dados (DDL)

```
/* -----
Tabela TIPOCLIENTE
----- */

CREATE TABLE TIPOCLIENTE (
  CODTIPOCLIENTE INTEGER NOT NULL,
  DESCRICAO VARCHAR(20),
  CREDITO FLOAT,
  PRIMARY KEY (CODTIPOCLIENTE)
);

CREATE GENERATOR TIPOCLIENTE_GEN;
SET TERM ^ ;
CREATE TRIGGER "TRIG_TIPOCLIENTE_ID" FOR "TIPOCLIENTE"
ACTIVE BEFORE INSERT POSITION 0
AS
begin
  IF (new.CODTIPOCLIENTE IS NULL) then
    begin
      new.CODTIPOCLIENTE = gen_id( TIPOCLIENTE_GEN, 1 );
    end
end
^
COMMIT WORK ^
SET TERM ;^

/* -----
Tabela CLIENTE
----- */

CREATE TABLE CLIENTE (
  CODCLIENTE INTEGER NOT NULL,
  CODTIPOCLIENTE INTEGER NOT NULL,
  NOME VARCHAR(70),
  CPF CHAR(11),
  RG VARCHAR(15),
  APELIDO VARCHAR(15),
  TELEFONE CHAR(10),
  CELULAR CHAR(10),
  LOGRADOURO VARCHAR(40),
  NUMERO VARCHAR(10),
  BAIRRO VARCHAR(20),
  CEP CHAR(8),
  CIDADE VARCHAR(30),
  ESTADO CHAR(2),
  DATANASCIMENTO DATE,
  SITUACAO CHAR(1),
  CREDITO FLOAT,
  PRIMARY KEY (CODCLIENTE),
  FOREIGN KEY (CODTIPOCLIENTE) REFERENCES TIPOCLIENTE (CODTIPOCLIENTE)
);

CREATE GENERATOR CLIENTE_GEN;
SET TERM ^ ;
CREATE TRIGGER "TRIG_CLIENTE_ID" FOR "CLIENTE"
ACTIVE BEFORE INSERT POSITION 0
```



```

AS
begin
  IF (new.CODCLIENTE IS NULL) then
    begin
      new.CODCLIENTE = gen_id( CLIENTE_GEN, 1 );
    end
  end
  ^
COMMIT WORK ^
SET TERM ;^

/* -----
Tabela VENDA
----- */

CREATE TABLE VENDA (
  CODVENDA INTEGER NOT NULL,
  CODCLIENTE INTEGER NOT NULL,
  DATAVENDA DATE,
  VALORTOTAL FLOAT,
  PRIMARY KEY (CODVENDA),
  FOREIGN KEY (CODCLIENTE) REFERENCES CLIENTE (CODCLIENTE)
);

CREATE GENERATOR VENDA_GEN;
SET TERM ^ ;
CREATE TRIGGER "TRIG_VENDA_ID" FOR "VENDA"
ACTIVE BEFORE INSERT POSITION 0
AS
begin
  IF (new.CODVENDA IS NULL) then
    begin
      new.CODVENDA = gen_id( VENDA_GEN, 1 );
    end
  end
end
^
COMMIT WORK ^
SET TERM ;^

/* -----
Tabela PRODUTO
----- */

CREATE TABLE PRODUTO (
  CODPRODUTO INTEGER NOT NULL,
  DESCRICAO VARCHAR (40),
  PRECO FLOAT,
  DATAVALIDADE DATE,
  DATACOMPRA DATE,
  PRIMARY KEY (CODPRODUTO)
);

CREATE GENERATOR PRODUTO_GEN;
SET TERM ^ ;
CREATE TRIGGER "TRIG_PRODUTO_ID" FOR "PRODUTO"
ACTIVE BEFORE INSERT POSITION 0
AS
begin
  IF (new.CODPRODUTO IS NULL) then
    begin
      new.CODPRODUTO = gen_id( PRODUTO_GEN, 1 );
    end
  end
end
^

```

```
COMMIT WORK ^
SET TERM ;^
```

```
/* -----
Tabela ITENSVENDA
----- */
```

```
CREATE TABLE ITENSVENDA (
  CODITEMVENDA INTEGER NOT NULL,
  CODVENDA INTEGER NOT NULL,
  CODPRODUTO INTEGER NOT NULL,
  QUANTIDADE INTEGER NOT NULL,
  VALORUNITARIO FLOAT NOT NULL,
  VALORPARCIAL FLOAT NOT NULL,
  PRIMARY KEY (CODITEMVENDA),
  FOREIGN KEY (CODVENDA) REFERENCES VENDA (CODVENDA),
  FOREIGN KEY (CODPRODUTO) REFERENCES PRODUTO (CODPRODUTO)
);
```

```
CREATE GENERATOR ITENSVENDA_GEN;
SET TERM ^ ;
CREATE TRIGGER "TRIG_ITENSVENDA_ID" FOR "ITENSVENDA"
ACTIVE BEFORE INSERT POSITION 0
AS
begin
  IF (new.CODITEMVENDA IS NULL) then
    begin
      new.CODITEMVENDA = gen_id( ITENSVENDA_GEN, 1 );
    end
  end
^
COMMIT WORK ^
SET TERM ;^
```

```
/* -----
Tabela CONTASRECEBER
----- */
```

```
CREATE TABLE CONTASRECEBER (
  CODCTARECEBER INTEGER NOT NULL,
  CODVENDA INTEGER NOT NULL,
  VALOR FLOAT NOT NULL,
  DATAVENCIMENTO DATE NOT NULL,
  DATAQUITACAO DATE,
  SITUACAO CHAR(1) NOT NULL,
  SALDO FLOAT,
  PRIMARY KEY (CODCTARECEBER),
  FOREIGN KEY (CODVENDA) REFERENCES VENDA (CODVENDA)
);
```

```
CREATE GENERATOR CONTASRECEBER_GEN;
SET TERM ^ ;
CREATE TRIGGER "TRIG_CONTASRECEBER_ID" FOR "CONTASRECEBER"
ACTIVE BEFORE INSERT POSITION 0
AS
begin
  IF (new.CODCTARECEBER IS NULL) then
    begin
      new.CODCTARECEBER = gen_id( CONTASRECEBER_GEN, 1 );
    end
  end
^
COMMIT WORK ^
SET TERM ;^
```

```

/* -----
Tabela FORMAPAGTO
----- */

CREATE TABLE FORMAPAGTO (
CODFORMAPAGTO INTEGER NOT NULL,
DESCRICAO VARCHAR(20),
PRIMARY KEY (CODFORMAPAGTO)
);

CREATE GENERATOR FORMAPAGTO_GEN;
SET TERM ^ ;
CREATE TRIGGER "TRIG_FORMAPAGTO_ID" FOR "FORMAPAGTO"
ACTIVE BEFORE INSERT POSITION 0
AS
begin
IF (new.CODFORMAPAGTO IS NULL) then
begin
new.CODFORMAPAGTO = gen_id( FORMAPAGTO_GEN, 1 );
end
end
^
COMMIT WORK ^
SET TERM ;^

/* -----
Tabela CONTASRECEBIDAS
----- */

CREATE TABLE CONTASRECEBIDAS (
CODCTARECEBIMENTO INTEGER NOT NULL,
CODCTARECEBER INTEGER NOT NULL,
CODFORMAPAGTO INTEGER NOT NULL,
VALORRECEBIDO FLOAT NOT NULL,
DATAPAGAMENTO DATE NOT NULL,
PRIMARY KEY (CODCTARECEBIMENTO),
FOREIGN KEY (CODCTARECEBER) REFERENCES CONTASRECEBER (CODCTARECEBER),
FOREIGN KEY (CODFORMAPAGTO) REFERENCES FORMAPAGTO (CODFORMAPAGTO)
);

CREATE GENERATOR CONTASRECEBIDAS_GEN;
SET TERM ^ ;
CREATE TRIGGER "TRIG_CONTASRECEBIDAS_ID" FOR "CONTASRECEBIDAS"
ACTIVE BEFORE INSERT POSITION 0
AS
begin
IF (new.CODCTARECEBIMENTO IS NULL) then
begin
new.CODCTARECEBIMENTO = gen_id( CONTASRECEBIDAS_GEN, 1 );
end
end
^
COMMIT WORK ^
SET TERM ;^

```

Inserção de Dados nas tabelas

```

/* TIPOCLIENTE */

INSERT INTO "TIPOCLIENTE" ("CODTIPOCLIENTE", "DESCRICAO",
"CREDITO") VALUES ( '1', 'ÓTIMO', '1000');

```

```

INSERT INTO "TIPOCLIENTE"      ("CODTIPOCLIENTE",      "DESCRICAO",
"CREDITO") VALUES (      '2',      'BOM',      '750');
INSERT INTO "TIPOCLIENTE"      ("CODTIPOCLIENTE",      "DESCRICAO",
"CREDITO") VALUES (      '3',      'REGULAR',      '400');
INSERT INTO "TIPOCLIENTE"      ("CODTIPOCLIENTE",      "DESCRICAO",
"CREDITO") VALUES (      '4',      'RUIM',      '0');
SET GENERATOR TIPOCLIENTE_GEN TO 4;

```

```

/* CLIENTE */

```

```

INSERT INTO "CLIENTE"      ("CODCLIENTE",      "CODTIPOCLIENTE",      "NOME",
"CPF",      "RG",      "APELIDO",      "TELEFONE",      "CELULAR",      "LOGRADOURO",
"NUMERO",      "BAIRRO",      "CEP",      "CIDADE",      "ESTADO",
"DATANASCIMENTO",      "SITUACAO",      "CREDITO") VALUES (      '1',      '2',
'RAFAEL DA SILVA GOULART',      '54681387346',      '3214 SSP BA',      'RAFAEL',
NULL,      NULL,      NULL,      NULL,      NULL,      NULL,      'BARREIRAS',      'BA',      '1972-
11-28',      '1',      NULL);
INSERT INTO "CLIENTE"      ("CODCLIENTE",      "CODTIPOCLIENTE",      "NOME",
"CPF",      "RG",      "APELIDO",      "TELEFONE",      "CELULAR",      "LOGRADOURO",
"NUMERO",      "BAIRRO",      "CEP",      "CIDADE",      "ESTADO",
"DATANASCIMENTO",      "SITUACAO",      "CREDITO") VALUES (      '2',      '3',
'FABIO FELICIO FIGUEIREDO',      '87643873438',      '1454 SSP BA',      'FABIO',
NULL,      NULL,      NULL,      NULL,      NULL,      NULL,      'IBOTIRAMA',      'BA',      '1979-
03-20',      '1',      NULL);
INSERT INTO "CLIENTE"      ("CODCLIENTE",      "CODTIPOCLIENTE",      "NOME",
"CPF",      "RG",      "APELIDO",      "TELEFONE",      "CELULAR",      "LOGRADOURO",
"NUMERO",      "BAIRRO",      "CEP",      "CIDADE",      "ESTADO",
"DATANASCIMENTO",      "SITUACAO",      "CREDITO") VALUES (      '3',      '1',
'DIVANILSON DE QUEIROZ RODRIGUES',      '00135467687',      '32135 SSP CE',
'DIVAN',      NULL,      NULL,      NULL,      NULL,      NULL,      NULL,      'LUIS EDUARDO
MAGALHAES',      'BA',      '1981-05-15',      '1',      NULL);
INSERT INTO "CLIENTE"      ("CODCLIENTE",      "CODTIPOCLIENTE",      "NOME",
"CPF",      "RG",      "APELIDO",      "TELEFONE",      "CELULAR",      "LOGRADOURO",
"NUMERO",      "BAIRRO",      "CEP",      "CIDADE",      "ESTADO",
"DATANASCIMENTO",      "SITUACAO",      "CREDITO") VALUES (      '4',      '4',
'LIMAYCON VIAN CARVALHO',      '65483213146',      '1114 SSP BA',      'BOI',      NULL,
NULL,      NULL,      NULL,      NULL,      NULL,      'BARREIRAS',      'BA',      '1985-08-01',
'0',      NULL);
INSERT INTO "CLIENTE"      ("CODCLIENTE",      "CODTIPOCLIENTE",      "NOME",
"CPF",      "RG",      "APELIDO",      "TELEFONE",      "CELULAR",      "LOGRADOURO",
"NUMERO",      "BAIRRO",      "CEP",      "CIDADE",      "ESTADO",
"DATANASCIMENTO",      "SITUACAO",      "CREDITO") VALUES (      '5',      '2',
'ROSA SILVA DE LIMA',      '88035468647',      '8734 SSP BA',      'ROSINHA',      NULL,
NULL,      NULL,      NULL,      NULL,      NULL,      'BARREIRAS',      'BA',      '1988-11-06',
'1',      NULL);
INSERT INTO "CLIENTE"      ("CODCLIENTE",      "CODTIPOCLIENTE",      "NOME",
"CPF",      "RG",      "APELIDO",      "TELEFONE",      "CELULAR",      "LOGRADOURO",
"NUMERO",      "BAIRRO",      "CEP",      "CIDADE",      "ESTADO",
"DATANASCIMENTO",      "SITUACAO",      "CREDITO") VALUES (      '6',      '1',
'MARILUCIA GOMES GUIMARÃES CARVALHO',      '00346879798',      '13578 SSP BA',
'MARILUCIA',      NULL,      NULL,      NULL,      NULL,      NULL,      NULL,      'SAO DESIDERIO',
'BA',      '1983-06-12',      '1',      NULL);
INSERT INTO "CLIENTE"      ("CODCLIENTE",      "CODTIPOCLIENTE",      "NOME",
"CPF",      "RG",      "APELIDO",      "TELEFONE",      "CELULAR",      "LOGRADOURO",
"NUMERO",      "BAIRRO",      "CEP",      "CIDADE",      "ESTADO",
"DATANASCIMENTO",      "SITUACAO",      "CREDITO") VALUES (      '7',      '4',
'TIAGO JORDÃO ROSSI MENSCH',      '03857876543',      '9834 SSP DF',      'TIAGO',
NULL,      NULL,      NULL,      NULL,      NULL,      NULL,      'SAO DESIDERIO',      'BA',
'1976-07-01',      '0',      NULL);
INSERT INTO "CLIENTE"      ("CODCLIENTE",      "CODTIPOCLIENTE",      "NOME",
"CPF",      "RG",      "APELIDO",      "TELEFONE",      "CELULAR",      "LOGRADOURO",
"NUMERO",      "BAIRRO",      "CEP",      "CIDADE",      "ESTADO",
"DATANASCIMENTO",      "SITUACAO",      "CREDITO") VALUES (      '8',      '3',
'FRANCO FLORIANO GONÇALVES AZEVEDO',      '32164987321',      '01245 SSP BA',
'FRANCO',      NULL,      NULL,      NULL,      NULL,      NULL,      NULL,      'LUIS EDUARDO

```

```

MAGALHAES',          'BA',      '1989-05-02',    '1',      NULL);
INSERT INTO "CLIENTE" ("CODCLIENTE",          "CODTIPOCLIENTE",          "NOME",
"CPF",      "RG",      "APELIDO",          "TELEFONE",          "CELULAR",          "LOGRADOURO",
"NUMERO",          "BAIRRO",          "CEP",      "CIDADE",          "ESTADO",
"DATANASCIMENTO",          "SITUACAO",          "CREDITO") VALUES (      '9',      '4',
'MANOEL VICENTE FRANCELINO FILHO',          '24687498749',      '1346546 SSP BA',
'MANEL',          NULL,      NULL,      NULL,      NULL,      NULL,      NULL,      'BARREIRAS',
'BA',      '1988-09-02',      '0',      NULL);
INSERT INTO "CLIENTE" ("CODCLIENTE",          "CODTIPOCLIENTE",          "NOME",
"CPF",      "RG",      "APELIDO",          "TELEFONE",          "CELULAR",          "LOGRADOURO",
"NUMERO",          "BAIRRO",          "CEP",      "CIDADE",          "ESTADO",
"DATANASCIMENTO",          "SITUACAO",          "CREDITO") VALUES (      '10',      '3',
'RICARDO LUIZ QUEIROZ CRUZ',          '38987134687',      '3132 SSP BA',      'BAHIA',
NULL,      NULL,      NULL,      NULL,      NULL,      NULL,      'LUIS EDUARDO MAGALHAES',
'BA',      '1975-10-03',      '1',      NULL);
SET GENERATOR CLIENTE_GEN TO 10;

```

/* PRODUTO */

```

INSERT INTO "PRODUTO" ("CODPRODUTO",          "DESCRICAO",          "PRECO",
"DATAVALIDADE",          "DATACOMPRA") VALUES (      '1',      'CELULAR',
'199,8999993896484',      NULL,      NULL);
INSERT INTO "PRODUTO" ("CODPRODUTO",          "DESCRICAO",          "PRECO",
"DATAVALIDADE",          "DATACOMPRA") VALUES (      '2',      'CARREGADOR CELULAR',
'25,1000003814697',      NULL,      NULL);
INSERT INTO "PRODUTO" ("CODPRODUTO",          "DESCRICAO",          "PRECO",
"DATAVALIDADE",          "DATACOMPRA") VALUES (      '3',      'MICROFONE',          '15',
NULL,      NULL);
INSERT INTO "PRODUTO" ("CODPRODUTO",          "DESCRICAO",          "PRECO",
"DATAVALIDADE",          "DATACOMPRA") VALUES (      '4',      'WEBCAM',          '85,5',
NULL,      NULL);
INSERT INTO "PRODUTO" ("CODPRODUTO",          "DESCRICAO",          "PRECO",
"DATAVALIDADE",          "DATACOMPRA") VALUES (      '5',      'DRIVE GRAVADOR CD-RW',
'120',      NULL,      NULL);
INSERT INTO "PRODUTO" ("CODPRODUTO",          "DESCRICAO",          "PRECO",
"DATAVALIDADE",          "DATACOMPRA") VALUES (      '6',      'DRIVE GRAVADOR COMBO
CD-RW / LEITOR DVD',          '150',      NULL,      NULL);
INSERT INTO "PRODUTO" ("CODPRODUTO",          "DESCRICAO",          "PRECO",
"DATAVALIDADE",          "DATACOMPRA") VALUES (      '7',      'DRIVE GRAVADOR DVD-RW /
CD-RW',          '210',      NULL,      NULL);
INSERT INTO "PRODUTO" ("CODPRODUTO",          "DESCRICAO",          "PRECO",
"DATAVALIDADE",          "DATACOMPRA") VALUES (      '8',      'MOUSE SERIAL',
'12',      NULL,      NULL);
INSERT INTO "PRODUTO" ("CODPRODUTO",          "DESCRICAO",          "PRECO",
"DATAVALIDADE",          "DATACOMPRA") VALUES (      '9',      'MOUSE PS2',          '17',
NULL,      NULL);
INSERT INTO "PRODUTO" ("CODPRODUTO",          "DESCRICAO",          "PRECO",
"DATAVALIDADE",          "DATACOMPRA") VALUES (      '10',      'MOUSE ÓTICO',          '20',
NULL,      NULL);
INSERT INTO "PRODUTO" ("CODPRODUTO",          "DESCRICAO",          "PRECO",
"DATAVALIDADE",          "DATACOMPRA") VALUES (      '11',      'TECLADO MULTIMIDIA',
'35',      NULL,      NULL);
INSERT INTO "PRODUTO" ("CODPRODUTO",          "DESCRICAO",          "PRECO",
"DATAVALIDADE",          "DATACOMPRA") VALUES (      '12',      'TECLADO PÉ DURO',
'20',      NULL,      NULL);
INSERT INTO "PRODUTO" ("CODPRODUTO",          "DESCRICAO",          "PRECO",
"DATAVALIDADE",          "DATACOMPRA") VALUES (      '13',      'SCANNER',          '150',
NULL,      NULL);
INSERT INTO "PRODUTO" ("CODPRODUTO",          "DESCRICAO",          "PRECO",
"DATAVALIDADE",          "DATACOMPRA") VALUES (      '14',      'MONITOR 15 POLEGADAS',
'300',      NULL,      NULL);
INSERT INTO "PRODUTO" ("CODPRODUTO",          "DESCRICAO",          "PRECO",
"DATAVALIDADE",          "DATACOMPRA") VALUES (      '15',      'MONITOR 17 POLEGADAS',
'420',      NULL,      NULL);
INSERT INTO "PRODUTO" ("CODPRODUTO",          "DESCRICAO",          "PRECO",
"DATAVALIDADE",          "DATACOMPRA") VALUES (      '16',      'MONITOR 15 POLEGADAS

```

```

LCD',      '750',  NULL,  NULL);
INSERT INTO "PRODUTO"      ("CODPRODUTO",      "DESCRICAO",      "PRECO",
"DATAVALIDADE",      "DATACOMPRA") VALUES (  '17',      'MONITOR 17 POLEGADAS
LCD',      '980',  NULL,  NULL);
SET GENERATOR PRODUTO_GEN TO 17;

```

```

/* VENDA */

```

```

INSERT INTO "VENDA"      ("CODVENDA",      "CODCLIENTE",      "DATAVENDA",
"VALORTOTAL") VALUES (  '1',      '3',      '2006-01-05',      NULL);
INSERT INTO "VENDA"      ("CODVENDA",      "CODCLIENTE",      "DATAVENDA",
"VALORTOTAL") VALUES (  '2',      '6',      '2006-01-07',      NULL);
INSERT INTO "VENDA"      ("CODVENDA",      "CODCLIENTE",      "DATAVENDA",
"VALORTOTAL") VALUES (  '3',      '1',      '2006-01-07',      NULL);
INSERT INTO "VENDA"      ("CODVENDA",      "CODCLIENTE",      "DATAVENDA",
"VALORTOTAL") VALUES (  '4',      '3',      '2006-01-10',      NULL);
INSERT INTO "VENDA"      ("CODVENDA",      "CODCLIENTE",      "DATAVENDA",
"VALORTOTAL") VALUES (  '5',      '10',      '2006-01-15',      NULL);
INSERT INTO "VENDA"      ("CODVENDA",      "CODCLIENTE",      "DATAVENDA",
"VALORTOTAL") VALUES (  '6',      '4',      '2006-01-20',      NULL);
INSERT INTO "VENDA"      ("CODVENDA",      "CODCLIENTE",      "DATAVENDA",
"VALORTOTAL") VALUES (  '7',      '5',      '2006-01-25',      NULL);
INSERT INTO "VENDA"      ("CODVENDA",      "CODCLIENTE",      "DATAVENDA",
"VALORTOTAL") VALUES (  '8',      '1',      '2006-01-28',      NULL);
INSERT INTO "VENDA"      ("CODVENDA",      "CODCLIENTE",      "DATAVENDA",
"VALORTOTAL") VALUES (  '9',      '8',      '2006-02-02',      NULL);

```