

1. INTRODUÇÃO À ARQUITETURA CLIENT/SERVER E DBX	2
PROGRAMAÇÃO CLIENTE/SERVIDOR	2
SGBDR	2
CONCEITOS DE BANCO DE DADOS	2
ARQUITETURA DO SGBDR	3
ARQUITETURA DELPHI 7 CLIENT/SERVER.....	3
DBEXPRESS E A ARQUITETURA DE ACESSO A DADOS	3
2. INTRODUÇÃO AO INTERBASE 6/7 E A LINGUAGEM SQL.....	4
CRIANDO O BANCO DE DADOS.....	4
CRIANDO TABELAS (DDL)	5
MANIPULANDO TABELAS (DML)	6
JUNÇÕES (JOINS)	8
INTEGRIDADE REFERENCIAL (FOREIGN KEYS).....	9
3. ACESSO AO IB COM DBEXPRESS NO DELPHI 7.....	12
COMPONENTES DBEXPRESS E DATA ACCESS	12
PROBLEMAS DO BDE.....	13
VANTAGENS DO DBEXPRESS	13
CURSORES UNIDIRECIONAIS	13
CRIANDO UMA APLICAÇÃO USANDO DBEXPRESS	14
CRIANDO O FORMULÁRIO DE ALUNOS	16
LOOKUPS	16
O DATASETPROVIDER	17
SQL MONITOR.....	18
PROVIDER FLAGS	19
CONSULTAS PARAMETRIZADAS.....	20
POR QUE SQLQUERY + DATASETPROVIDER + CLIENTDATASET ?	21
SIMPLEDATASET E SQLCLIENTDATASET	22
CONSULTAS PARAMETRIZADAS E PROCURAS.....	22
JOINS AO INVÉS DE LOOKUPS	24
VIEWS	26
TRIGGERS E EXCEPTIONS	26
TRANSAÇÕES EM DBEXPRESS	27
STORED PROCEDURES.....	27
4. RELATÓRIOS COM QUICKREPORT USANDO SQL	28
INSTALANDO O QUICKREPORT NO DELPHI 7	28
CRIANDO A CONSULTA.....	28
RELATÓRIO SIMPLES.....	29
SUMÁRIOS.....	30
RELATÓRIOS AGRUPADOS	30
5. GRÁFICOS COM DBCHAT USANDO SQL.....	31
6. XML COM DELPHI 7	32
SINTAXE BÁSICA DA XML	32
DOCUMENTOS XML	33
CLIENTDATASET E XML DATAPACKET.....	33
XML MAPPER	34
USANDO DOCUMENTOS XML COMO BANCO DE DADOS	35
APLICAÇÕES B2B - TRANSFERÊNCIA DE DADOS E SERVIÇOS USANDO XML.....	36
RETIRANDO DADOS DO INTERBASE E DISTRIBUINDO EM FORMATO XML (B2B).....	37

1.Introdução à Arquitetura Client/Server e DBX

[g1] Comentário: To-Do: trocar nomes dos componentes para ficar no padrão do SI*

Programação cliente/servidor

- Antigamente os computadores trabalhavam isoladamente, com poucas informações;
- Com o crescimento das empresas cada setor possuía seu próprio computador;
- Surgiu a necessidade de compartilhar as informações;
- Surgiu a programação multi-usuário, somente com arquivos físicos compartilhados, sem um servidor;
- A Programação cliente/servidor introduziu os SGBDR;
- Os dados ficam separados do código de interface cliente;

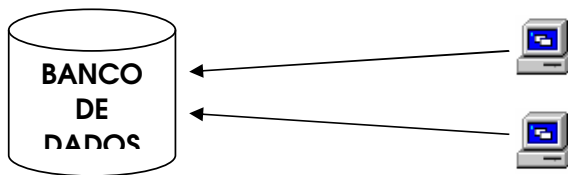


Figura. Arquitetura de cliente/servidor

SGBDR

- Sistema Gerenciador de Banco de Dados Relacional (*DataBase Manager System*);
- Responsável pelo gerenciamento das informações;
- Exemplos: *Microsoft SQL Server, Oracle, DB2 (IBM) Interbase (Borland), etc*;
- Serve dados, enquanto sistemas locais só gerenciam arquivos;
- Permite a independência entre dados e programas;
- Permite o controle de redundância de dados, através de chaves, relacionamentos, triggers, regras, etc;
- Permite o controle de acessos através do uso de senhas, usuários, permissões;
- Garante a integridade dos dados;
- Permite a facilidade de criação de novas aplicações;
- Controle automático de relacionamento entre registros (*Foreign Keys*);
- Bloqueio Otimista ao invés de Pessimista;
- Transações Atômicas;
- Linguagem SQL;
- Suporte a Backup e Restore;
- Espelhamento;
- Propriedades ACID – Atomicidade, Consistência, Isolamento e Durabilidade (Transações);

Conceitos de Banco de Dados

- Gerenciado pelo SGBDR;
- Conjunto de Tabelas;
- Tabelas têm campos
- As Tabelas se relacionam (*Foreign Keys*);
- Views (visões);
- Índices – Chaves Primárias;
- Permissões em nível de tabela – só para inserir, atualizar ou consultar;

Arquitetura do SGBDR

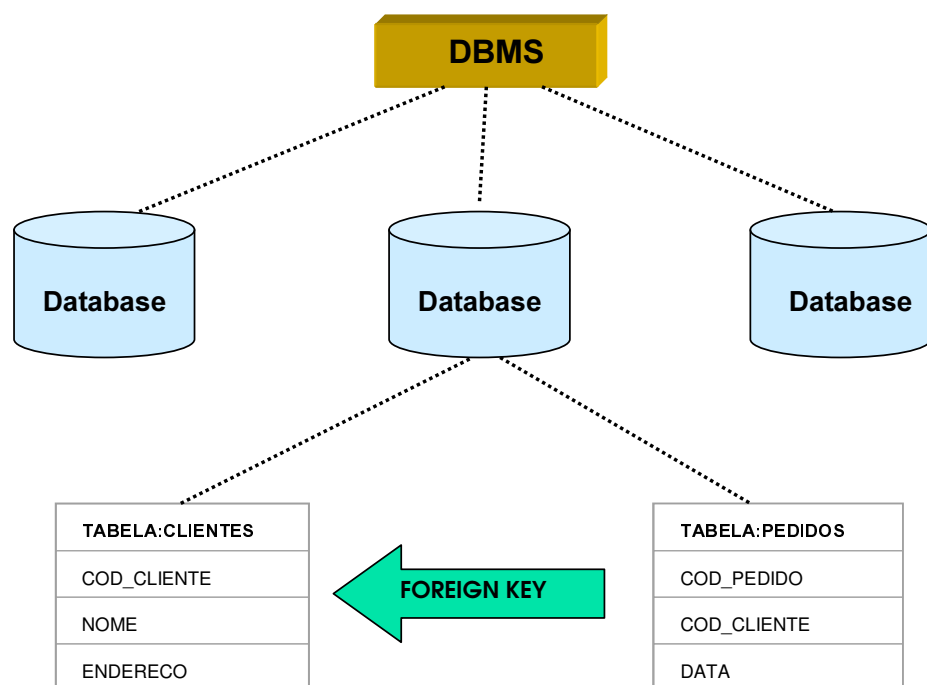


Figura. Arquitetura de um SGBDR tradicional, como Bancos de Dados, Tabelas e Relacionamentos

Arquitetura Delphi 7 Client/Server

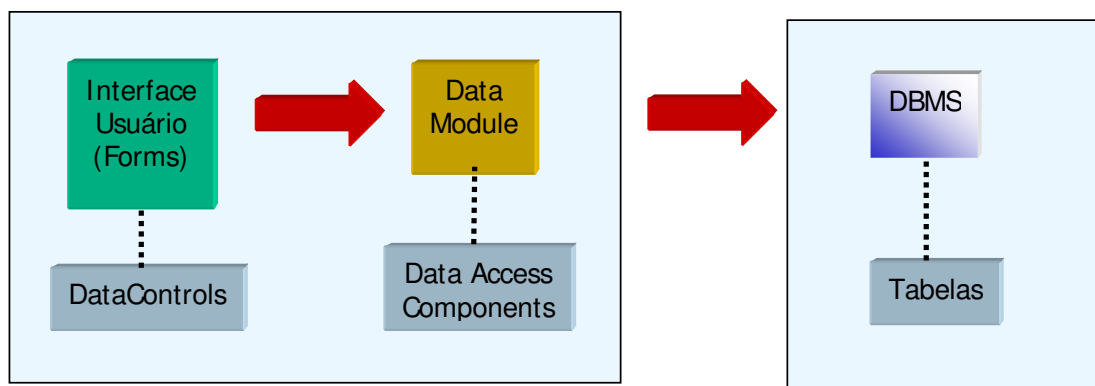


Figura. Modelo de Acesso a Dados tradicional no Delphi

dbExpress e a Arquitetura de Acesso a Dados

O dbExpress é uma arquitetura de acesso a dados muito poderosa que foi introduzida no Delphi 6 e no Kylix. O dbExpress dispensa o uso do BDE para o acesso a dados e realiza um acesso muito mais rápido e leve. A configuração, a instalação e a distribuição são também muito mais fáceis.

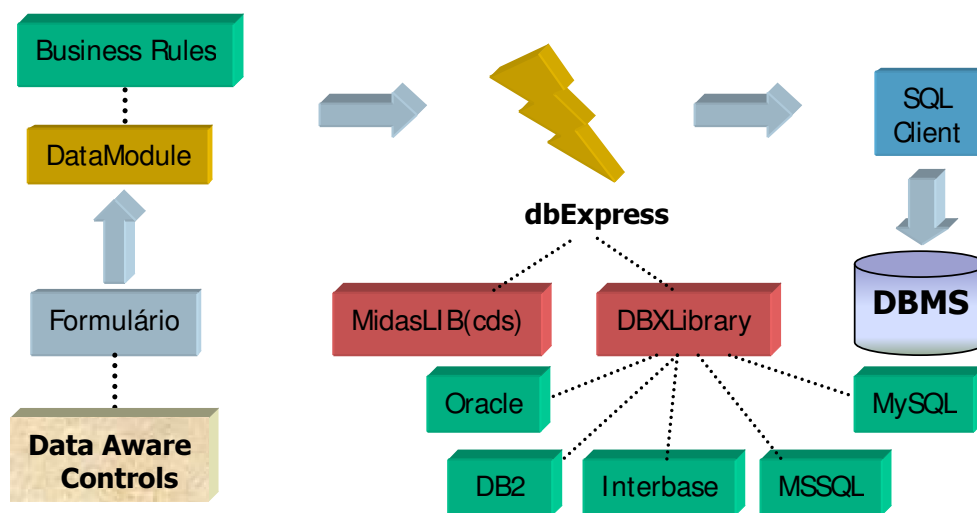


Figura. Arquitetura dbExpress de acesso a dados

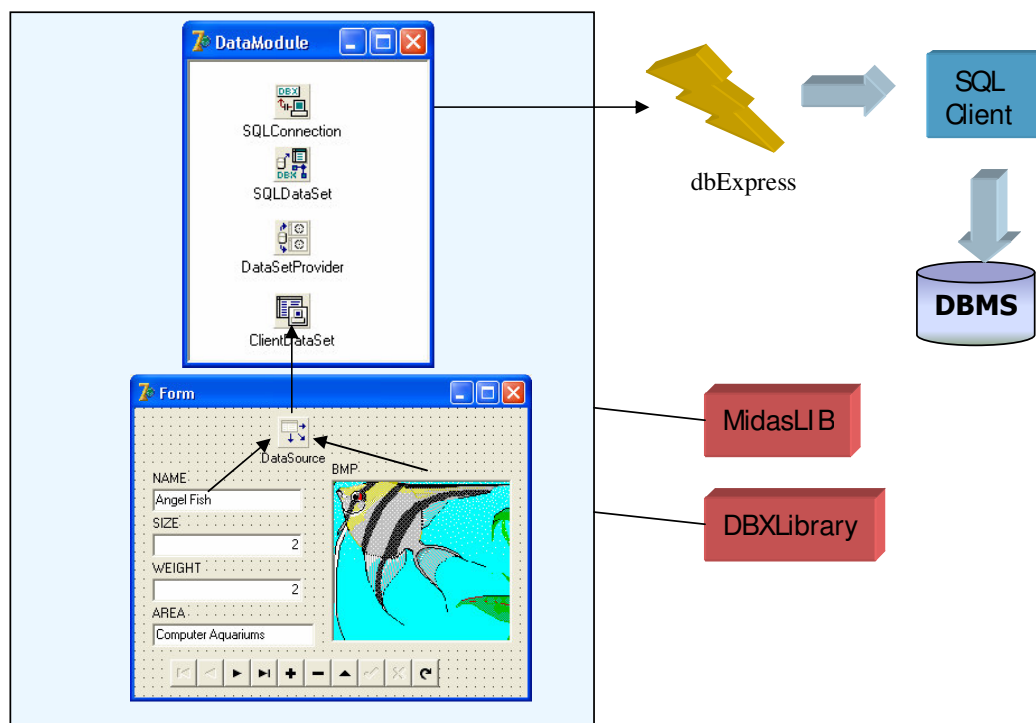


Figura. Arquitetura dbExpress

2. Introdução ao Interbase 6/7 e a linguagem SQL

Criando o Banco de Dados

Inicie o servidor local no *IBConsole*, dando um duplo clique no item Local Server e fornecendo as credenciais do banco de dados (*User name = SYSDBA e Password = masterkey*). Dê um clique de direita sobre o item Databases e escolha a opção *Create Database*, como mostra a figura a seguir:

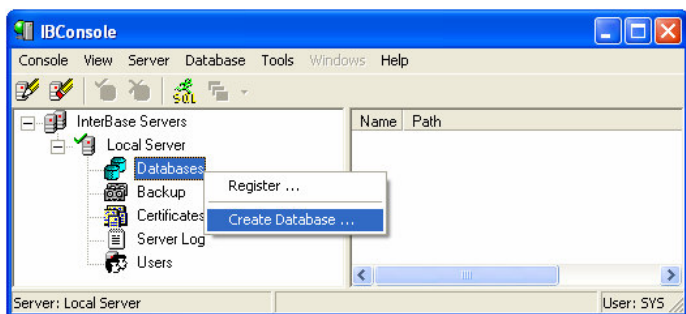


Figura. Criando um banco de dados no Interbase

Digite o caminho completo e nome do arquivo GDB do banco de dados em *FileName*, altere o item *Page Size* para 2048, *Default Character Set* para WIN1252, *SQL* para *Dialect 3* e *Alias* para ESCOLA.

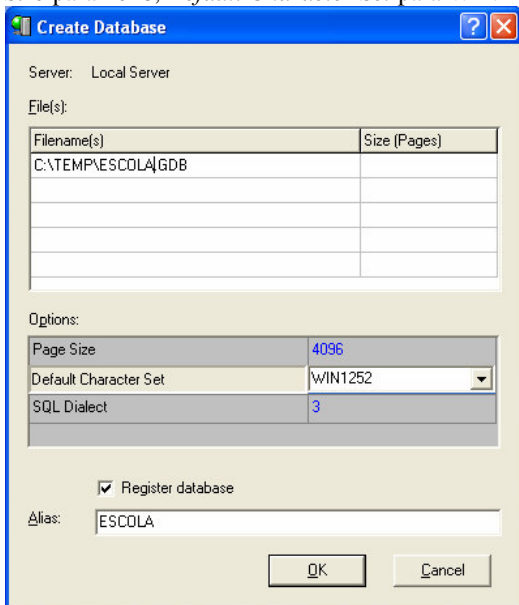


Figura. Parâmetros do novo banco de dados

Criando Tabelas (DDL)

No *IBConsole*, clique no botão *InteractiveSQL*. Este utilitário é utilizado para passar comandos SQL para o servidor *Interbase*. Para criar uma tabela de alunos digite o comando como mostra a figura a seguir.

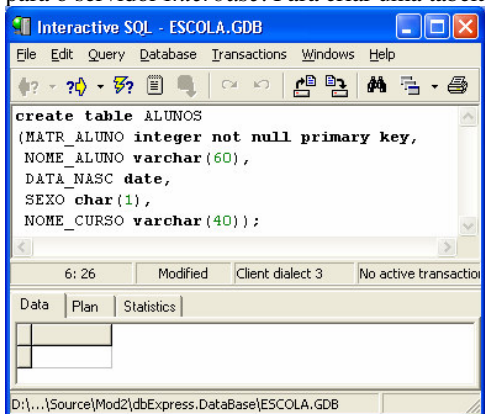



Figura. Criando uma tabela

ATENÇÃO:

Use exatamente o nome dos campos mostrados na figura ao lado, em letras maiúsculas. Preserve também o nome de tabela.

```
create table ALUNOS
(MATR_ALUNO integer not null primary key,
NOME_ALUNO varchar(60),
DATA_NASC date,
SEXO char(1),
NOME_CURSO varchar(40));
```

Pressione CTRL+E para executar a instrução ou aperte o botão *Execute Query* . Faça isso sempre depois que digitar um comando e quiser executá-lo.

Observe que o campo MATR_ALUNO não pode ser nulo (*not null*), já que é chave-primária (primary key). Propositalmente criamos o campo NOME_CURSO como char. Este campo na verdade deveria estar em uma tabela separada, chamada CURSOS. O ALUNO deve conter apenas uma referência para um determinado CURSO, através de um campo *integer* chamado COD_CURSO.

Atenção: para fins didáticos criamos aqui apenas duas tabelas. Em uma situação real haveria a necessidade de ser criar mais uma tabela, que relacione CURSOS e ALUNOS (n para n), já que um aluno pode fazer mais de um curso.

Para excluir o campo NOME_CURSO digite o comando abaixo.

```
alter table ALUNOS
drop NOME_CURSO
```

Caso quiséssemos excluir toda a tabela digitaríamos o comando abaixo (**apenas observe, não digite**):

```
drop table ALUNOS
```

Para adicionar o novo campo digite:

```
alter table ALUNOS
add COD_CURSO integer
```

Para criar a tabela de CURSOS digite o comando abaixo:

```
create table CURSOS
(COD_CURSO integer not null primary key,
NOME_CURSO varchar(40))
```

Manipulando Tabelas (DML)

Para inserir dados na tabela alunos use o comando a seguir (ainda no *Interactive SQL*):

```
insert into ALUNOS values
(1, 'GUINHTER', '01/01/1940', 'M', 0);
```

Observe que os valores devem ser digitados na ordem exata em que os campos foram criados, e nenhum valor para um determinado campo pode ser omitido.

Para inserir registros sem especificar todos os campos, ou ainda em uma ordem diferente, basta você especificar o nome e a ordem dos campos que irá inserir:

```
insert into ALUNOS (MATR_ALUNO, NOME_ALUNO, SEXO)
values (2, 'RUDOLFO', 'M')
```

Tarefa: Insira vários registros (10 a 20) usando as sintaxes mostradas acima. Dê um código de 1 a 3 para os cursos dos alunos. Cadastraremos os nomes dos cursos depois na tabela CURSOS.

Para alterar o nome de um determinado aluno, o qual já tenha sido inserido na tabela do banco de dados, faça como no exemplo a seguir, onde altero o NOME e o COD_CURSO do aluno onde MATR_ALUNO é 1.

```
update ALUNOS
set
  NOME_ALUNO='GUINHTER PAULI',
  COD_CURSO=2
where
  MATR_ALUNO=1
```

Caso a cláusula WHERE não seja utilizada, todos os registros serão atualizados. Por exemplo, o código abaixo altera o valor do campo COD_CURSO de todos os alunos para 1 (**apenas observe, não digite**):

```
update ALUNOS
set
  COD_CURSO=1
```

Para apagar um determinado registro utilize o comando DELETE (se você não especificar uma cláusula WHERE todos os registros serão apagados):

```
delete
  from ALUNOS
where
  MATR_ALUNO=2
```

O comando SELECT serve para retornar (buscar) informações de um banco de dados. Por exemplo, para visualizar todos os registros da tabela ALUNOS utilizamos o comando SELECT *:

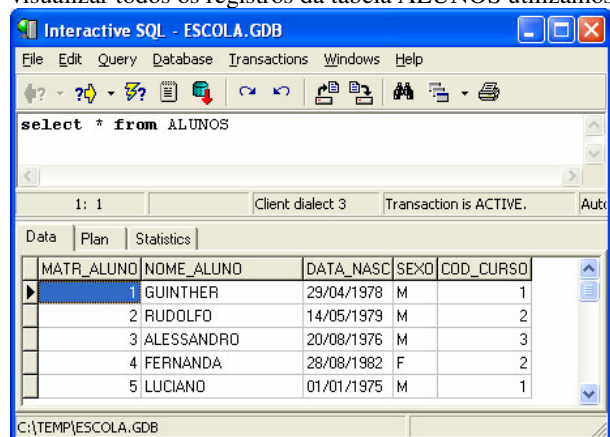


Figura. Comando SELECT

Para consultar apenas o campo MATR_ALUNO e NOME_ALUNO da tabela ALUNOS, digite o comando abaixo:

```
select MATR_ALUNO, NOME_ALUNO
from ALUNOS
order by NOME_ALUNO
```

Para consultar apenas os alunos do onde COD_CURSO for 1 digite o comando abaixo:

```
select MATR_ALUNO, NOME_ALUNO
from ALUNOS
where COD_CURSO=1
order by NOME_ALUNO
```

Para sabe quantos alunos estão cadastrados utilize o comando abaixo:

```
select count(MATR_ALUNO)
from ALUNOS
```

Agora insira 3 cursos na tabela CURSOS, usando o comando INSERT.

```
insert into CURSOS values (1,'DELPHI');
insert into CURSOS values (2,'KYLIX');
insert into CURSOS values (3,'E-COMMERCE');
```

Depois que você estiver familiarizado com os comandos de DML, poderá utilizar as ferramentas visuais do *IBConsole* para manipular dados. Na janela principal do *IBConsole* vá até o Item *Tables* e dê um duplo clique na tabela de *ALUNOS*. Observe todas as guias disponíveis na janela, e vá até o item *Data*.

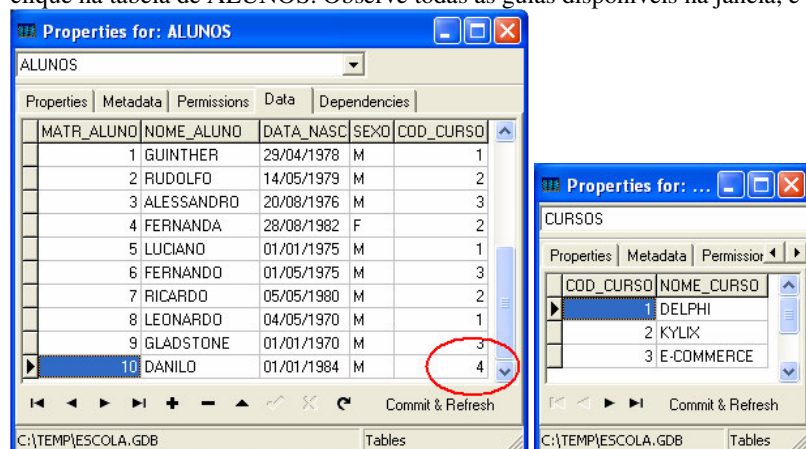


Figura. Manipulando os dados no IBConsole

Nesta janela você pode alterar os dados, excluir registros, inserir, etc. Depois que tiver feito todas as operações, lembre-se de apertar o botão *Commit and Refresh*.

Agora faça o seguinte: Deixe somente os 3 cursos cadastrados na tabela *CURSOS* como mostra a figura. Certifique-se que todos os Alunos possuem o *COD_CURSO* entre 1 e 3, de forma variada. Deixe apenas um único aluno com um *COD_CURSO* inválido, como 4 por exemplo. O código 4 não existe na tabela *CURSOS*. Isso caracteriza uma **inconsistência**, que vamos corrigir mais tarde.

Junções (JOINS)

No *IBConsole*, volte ao *Interactive SQL* . Agora dê o seguinte comando de consulta:

```
select MATR_ALUNO, NOME_ALUNO, COD_CURSO
from ALUNOS
```

Como você está lembrado, este comando retorna os dados de uma tabela. E se eu quisesse mostrar nesta mesma consulta o nome do curso (*NOME_CURSO*) que está na outra tabela, e não apenas o código do curso do aluno (*COD_CURSO*)?

Para isso você pode utilizar uma junção, ou Join, que une dados entre tabelas através de campos de ligação, neste caso *COD_CURSO* da tabela *CURSOS* com *COD_CURSO* da tabela *ALUNOS*. Digite o seguinte comando:

```
select
  A.MATR_ALUNO,
  A.NOME_ALUNO,
  A.COD_CURSO,
  C.NOME_CURSO
from
  ALUNOS A,
  CURSOS C
where
  A.COD_CURSO=C.COD_CURSO
order by
  A.NOME_ALUNO
```

A junção acima por padrão é um *INNER JOIN*, e equivale a instrução abaixo:


```
select
  A.MATR_ALUNO,
  A.NOME_ALUNO,
  A.COD_CURSO,
  C.NOME_CURSO
from
```

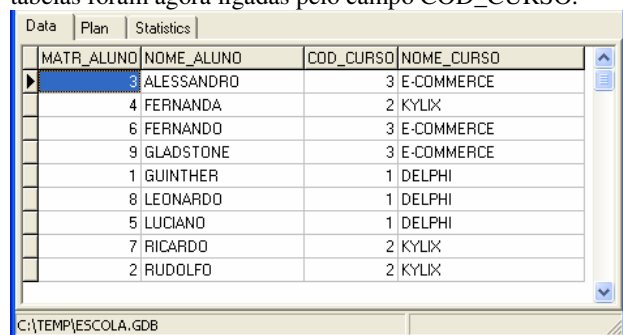


```

ALUNOS A
inner join
CURSOS C
on
A.COD_CURSO=C.COD_CURSO
order by
A.NOME_ALUNO

```

Pressione CTRL+E para executar a instrução ou aperte o botão *Execute Query* . Observe que as duas tabelas foram agora ligadas pelo campo COD_CURSO.



MATR_ALUNO	NOME_ALUNO	COD_CURSO	NOME_CURSO
3	ALESSANDRO	3	E-COMMERCE
4	FERNANDA	2	KYLIX
6	FERNANDO	3	E-COMMERCE
9	GLADSTONE	3	E-COMMERCE
1	GUINTHER	1	DELPHI
8	LEONARDO	1	DELPHI
5	LUCIANO	1	DELPHI
7	RICARDO	2	KYLIX
2	RUDOLFO	2	KYLIX

Figura. Consulta usando JOIN simples

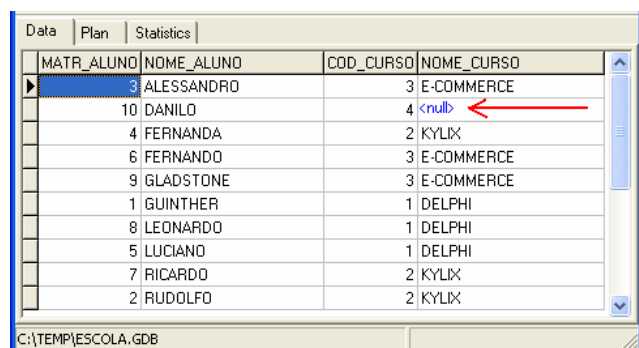
Mas se você observar bem verá que aquele aluno que possuía o código de curso “4” não está no resultado final. Isto acontece porque, no momento da montagem da consulta, o “**join**” selecionou este aluno e não encontrou uma equivalência para o seu curso na tabela CURSOS. Logo, o registro inteiro é eliminado do resultado final da consulta.

O comando abaixo elimina este problema, trazendo mesmo assim este aluno. O campo do curso então é preenchido por um valor NULO.

```

select
A.MATR_ALUNO,
A.NOME_ALUNO,
A.COD_CURSO,
C.NOME_CURSO
from
ALUNOS A
left join
CURSOS C
on
A.COD_CURSO=C.COD_CURSO
order by
A.NOME_ALUNO

```




MATR_ALUNO	NOME_ALUNO	COD_CURSO	NOME_CURSO
3	ALESSANDRO	3	E-COMMERCE
10	DANILO	4	<null>
4	FERNANDA	2	KYLIX
6	FERNANDO	3	E-COMMERCE
9	GLADSTONE	3	E-COMMERCE
1	GUINTHER	1	DELPHI
8	LEONARDO	1	DELPHI
5	LUCIANO	1	DELPHI
7	RICARDO	2	KYLIX
2	RUDOLFO	2	KYLIX

Figura. Consulta usando LEFT JOIN

Integridade Referencial (Foreign Keys)

Até agora nada do que fizemos diz ao banco de dados que as tabelas estão relacionadas. Tudo o que existe entre as duas tabelas é um apenas o “nome” de um campo em comum. Para criar uma integridade física entre essas duas tabelas, através do campo COD_CURSO, usaremos *Foreign Keys*.

Uma *Foreign Key* pode prevenir que um aluno seja matriculado em um curso que não existe. E também que você exclua da tabela um registro de curso enquanto alunos estiverem matriculados naquele curso. Caso contrário aqueles alunos também possuiriam códigos de curso inválidos.

Para criar esta relação (*Foreign Key*), no *IBConsole*, volte ao *Interactive SQL* .

Atenção: Uma integridade deve ser criada no mesmo momento que criamos as tabelas, quando ainda estão vazias e, portanto consistentes.

Como já temos dados cadastrados, você primeiro deve remover a inconsistência entre as tabelas, no caso o aluno que possui o código de curso inválido. Altere então seu `COD_CURSO` de 4 para 3. Se quiséssemos excluir todos os alunos que não possuem correspondência de `COD_CURSO` na tabela `CURSOS`, faríamos o seguinte código (**apenas observe**).

```
delete
from ALUNOS
where COD_CURSO not in
(select COD_CURSO from CURSOS)
```

Em um sistema real logicamente você não vai querer perder os dados dos alunos que possuem inconsistência nos dados. Você poderia criar um curso na tabela `CURSOS` chamado “NULO”, como `COD_CURSO` sendo 0:

```
insert into CURSOS values (0,'NULO');
```

Depois faça o seguinte para dar código 0 ao `COD_CURSO` de todos os alunos que possuem `COD_CURSO` inconsistente:

```
update ALUNOS
set COD_CURSO=0
where
COD_CURSO not in
(select COD_CURSO from CURSOS)
```

Se você verificar notará que o aluno que possuía código de curso 4 agora possui 0. Pronto! Todos os alunos possuem uma correspondência na tabela `CURSOS`. Então podemos criar a integridade entre as tabelas através da *Foreign Key*, usando o seguinte comando:

```
alter table ALUNOS
add constraint FK_ALUNOS_CURSOS
foreign key (COD_CURSO)
references CURSOS (COD_CURSO)
on delete no action
on update cascade
```

Agora digite o seguinte código para “tentar” incluir um aluno com um código de curso inválido:

```
insert into ALUNOS (MATR_ALUNO,NOME_ALUNO,COD_CURSO)
values (20,'CUJO',4);
```

Como você pode notar, o curso 4 não existe. Você deve receber um aviso de violação de integridade como mostrado a seguir:



Figura. Foreign Key em ação: Aluno não pode ser incluído com curso inválido

Agora faça outra tentativa que poderia tirar a consistência do banco, como excluir o curso 1. Também não será possível, porque existem alunos matriculados neste curso.

```
delete from CURSOS
where COD_CURSO=1
```

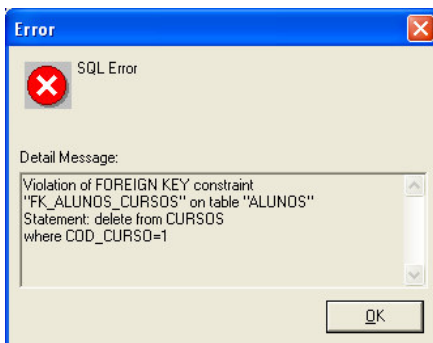


Figura. Foreign Key em ação: Um curso não pode ser excluído se alunos estiverem matriculados nele

Agora faça outro teste, altere o código do Curso 1 para 10.

```
update CURSOS
set COD_CURSO=10
where COD_CURSO=1
```

E por incrível que pareça se você executar este comando nada de estranho acontecerá! A operação será executada normalmente! Mas se o curso 1 passou a ser 10, o que aconteceu com os alunos que estavam matriculados no Curso 1? Ficaram inconsistentes? E a integridade? Dê um *select* na tabela de Alunos e repare que no momento que o Curso 1 passou a 10 todos os alunos também passaram a reconhecer o 10 como novo código:

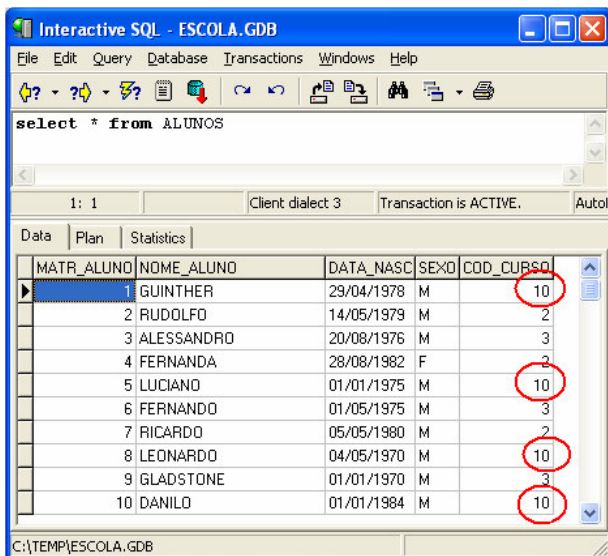


Figura. Cascade Update: todos os alunos receberam o novo código da tabela relacionada.

Este tipo de comportamento só foi possível porque no momento da criação da *Foreign Key* especificamos que, no caso de um *Update*, o servidor deveria fazer uma “atualização em cascata” nas tabelas relacionadas, veja:

```
on update cascade
```

Você notou que quando tentamos excluir um curso, a *Foreign Key* detectou que haviam registros relacionados na tabela alunos e abortou a operação. Se por acaso você quisesse realizar uma “exclusão em cascata” (excluir todos os alunos de um curso quando o curso fosse excluído), bastava ter criado a *Foreign Key* com a seguinte sintaxe no item *on delete*:

```
on delete cascade
```

Vale observar aqui que a Integridade é “**Server-Side**”. Você não precisará ficar testando a nível de camada de apresentação (no seu software em Delphi) se há ou não relação ou integridade entre as tabelas, se determinado registro pode ou não ser excluído. **Isso é feito no servidor!**

3. Acesso ao IB com dbExpress no Delphi 7

Componentes dbExpress e Data Access

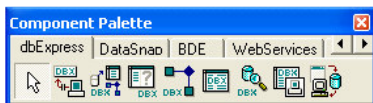












Figura. Componentes dbExpress de acesso a dados.

Componente	Função
 SQLConnection	Componente encarregado de conectar ao servidor SQL. Substitui o antigo <i>TDataBase</i> (BDE)
 SQLQuery	Recupera dados do servidor usando uma instrução SQL. Substitui a antiga <i>TQuery</i> (BDE).
 SQLStoredProc	Executa um Stored Procedure no servidor. Substitui o antigo <i>TStoredProc</i> (BDE).
 SQLTable	Acessa e recupera dados de uma tabela no servidor. Substitui a antiga <i>TTable</i> (BDE).

 SQLDataSet	Pode atuar como um Query, Table ou Stored Procedure. Não possui equivalente no BDE.
 SQLMonitor	Faz o trace das mensagens entre servidor e cliente SQL. Substitui o SQL Monitor (BDE)
 SQLClientDataSet	Equivale ao <i>ClientDataSet</i> + <i>DataSetProvider</i> + <i>SQLDataSet</i> (3 em 1). Surgiu o morreu no Delphi 6
 SimpleDataSet	Equivale ao <i>ClientDataSet</i> + <i>DataSetProvider</i> + <i>SQLDataSet</i> + <i>SQLConnection</i> (4 em 1)
 ClientDataSet	Faz a cache dos dados recuperados pelo dbExpress
 DataSetProvider	Empacota dados e resolve atualizações vindas do delta do <i>ClientDataSet</i>

Problemas do BDE

Abaixo destaco alguns dos principais problemas encontrados ao se trabalhar com o BDE, bem como algumas de suas limitações:

- Mecanismo de cache de dados rudimentar e problemático;
- Necessidade de distribuição de várias bibliotecas;
- Requer configuração adicional;
- Não roda no Linux, ou seja, não é cross-platform;
- Pouco controle sobre o tráfego de dados;
- Consumo elevado de recursos;
- Dificulta o desenvolvimento de novos drivers de acesso;
- O BDE foi decretado “morto” pela Borland e não será mais incluído no Delphi a partir da versão 8;

Vantagens do dbExpress

- Utilização mínima de recursos de sistema;
- Elevada performance e velocidade;
- Desenvolvimento cross-platform (o Kylix e C++ também possuem dbExpress, além do Delphi for .NET);
- Distribuição simples e descomplicada; (a biblioteca é indicada na propriedade *LibraryName* do *SQLConnection*).
- Fácil desenvolvimento de novos drivers;
- Maior controle sobre o tráfego de dados na rede;
- Limitação: não trabalha com bancos locais (paradox, Dbase);

Cursors Unidirecionais

Ao contrário do BDE, o dbExpress **não mantém em memória** qualquer informação recuperada pelo cursor SQL. Isso porque o dbExpress trabalha com o conceito de cursors unidirecionais. Dessa forma, algumas operações efetuadas antigamente sobre um componente *TQuery* de consulta não são permitidos sobre os componentes de consulta do dbExpress (por exemplo o *TSQLQuery*). Algumas das limitações dos datasets do dbExpress são:



- * Não é possível fazer uma localização com *Locate*, *FindNearest* ou *FindKey*;
- * Não é possível usar filtros com *Filter*, *OnFilterRecord* ou *SetRange*;
- * Não é permitido criar campos do tipo *Lookup*;
- * Não é permitido fazer edição ou alteração dos dados recuperados pelo cursor;
- * A navegação é limitada aos comandos *First* e *Next*, não sendo permitido chamar *Last* ou *Prior*;

Mas de que adianta ser rápido, leve e simples se não podemos realizar a maioria das operações sobre os *DataSets*? A resposta está em um recurso já muito conhecido na tecnologia MIDAS / DataSnap e que agora foi adotado oficialmente pelo dbExpress, o *ClientDataSet*.

Criando uma aplicação usando dbExpress

Crie uma nova aplicação no Delphi 7. Clique em *File|Save All*. Salve o formulário com o nome de “uFrmAluno” e o projeto como “Escola.DPR”. Dê o nome de “FrmAlunos” ao formulário.

Clique em *File|New|DataModule*. Salve o *DataModule* com o nome de “uDM” e configure sua propriedade *Name* para “DM”.

Coloque no *DataModule* um componente *SQLConnection* . Dê um duplo clique neste componente para abrir o editor de conexões do *dbExpress*. Clique no botão *Add Connection*  para adicionar uma nova conexão. A tela abaixo será mostrada, onde você deve escolher o *Driver Name Interbase*. Digite ESCOLA para *Connection Name* (Alias da conexão) e pressione OK.

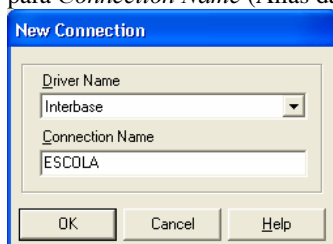


Figura. Adicionando uma conexão ao Interbase

Voltando ao editor de conexões, digite o caminho completo do arquivo de banco de dados na propriedade *Database* (coloque o nome ou IP do servidor na frente do caminho). Observe o usuário e senha padrão do Interbase já preenchidos. Verifique se o *SQLDialect* está configurado como “3” e *ServerCharSet* como *WIN1252*. Aperte OK.

Atenção: sempre use o formato “IP:\diretorio\banco.gdb” para o nome do database quando usar Interbase.

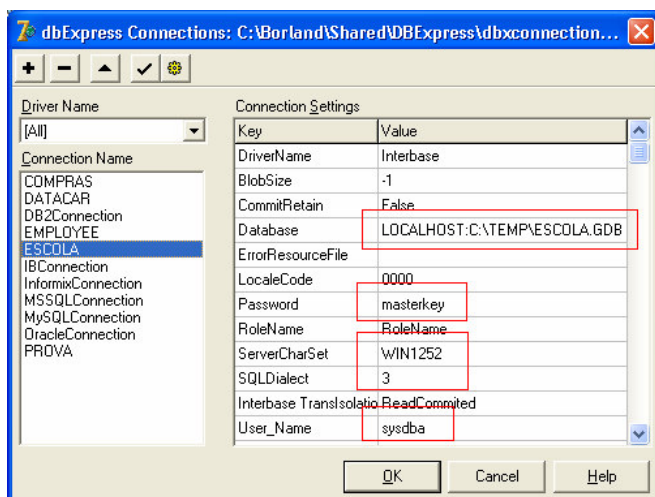


Figura. Parâmetros da conexão Interbase usando dbExpress

Configure a propriedade *LoginPrompt* do *SQLConnection* para *False*, e *Connected* para *True*. Coloque no *DataModule* duas *SQLQueries*, dois *DataSetproviders* e dois *ClientDatasets*.

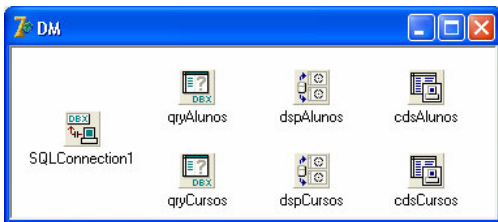


Figura. Trio de componentes (SQLQuery+DataSetProvider+ClientDataSet) para acessar uma tabela no Interbase

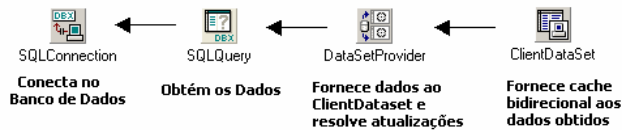


Figura. SQLConnection e trio de componentes (SQLQuery+DSP+CDS) para acessar uma tabela no Interbase

Configure as propriedades desses componentes como mostrado a seguir:



qryAlunos (SQLQuery de Alunos)
 Name = qryAlunos
 SQLConnection = SQLConnection1
 SQL =

```
select * from ALUNOS order by NOME_ALUNO
```



dspAlunos (DataSetProvider de Alunos)
 Name = dspAlunos
 DataSet = qryAlunos



cdsAlunos (ClientDataset de Alunos)
 Name = cdsAlunos
 ProviderName = dspAlunos
 Active = True



qryCursos (SQLQuery de Cursos)
 Name = qryCursos
 SQLConnection = SQLConnection1
 SQL =

```
select * from CURSOS order by NOME_CURSO
```



dspCursos (DataSetProvider de Cursos)
 Name = dspCursos
 DataSet = qryCursos



```
cdsCursos(ClientDataset de Cursos)
Name = cdsCursos
ProviderName = dspCursos
Active = True
```

Criando o formulário de Alunos

Dê um duplo clique no *ClientDataset* de Alunos (*cdsAlunos*). No editor de campos de um clique de direita e escolha a opção *Add All Fields*. Selecione todos os *TFields* e arraste para o formulário de alunos. O Delphi pergunta se você quer usar o DM a partir do formulário. Responda que sim.

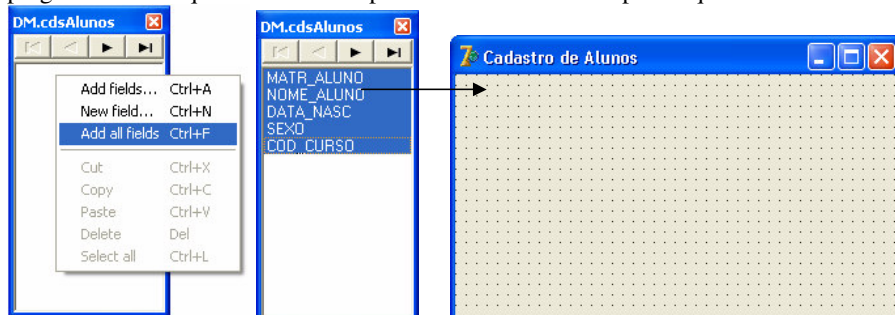


Figura. Adicionando TFields e criando DataControls no formulário de Alunos.

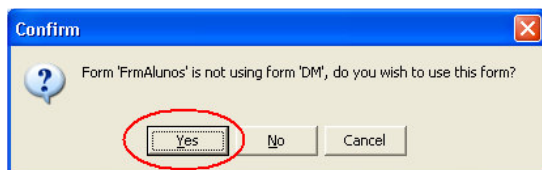


Figura. FormAalunos não está usando o DM, você quer adicioná-lo à cláusula uses deste formulário?

`uses uDM;` ← isso é adicionado a unit do formulário quando você responde sim a pergunta acima

Coloque no formulário um componente *DBNavigator* da guia *DataControls* e configure sua propriedade *DataSource*. Coloque um *BitBtn* (ou qualquer outro botão) no formulário, e dê o *Caption* de “Aplicar Cache”. Dê um duplo clique nele e digite no seu evento *OnClick* o seguinte:

```
dm.cdsAlunos.ApplyUpdates(0);
```

Execute a aplicação pressionando F9 e teste inclusões, alterações, exclusões, navegação, etc.



Figura. Formulário de Alunos com os DataControls e o DataSource

Lookups

Como posso mostrar o Nome do Curso ao lado do código do curso no formulário anterior?

Para isso podemos construir um campo *Lookup* no *ClientDataset* exatamente da mesma forma como fazemos em aplicações locais com *ClientDataSet*.

No *DataModule*, selecione o *ClientDataSet* de Alunos (*cdsAlunos*). **Atribua sua propriedade *Active* para *False***. Dê um duplo clique nele, e dê um clique de direita no editor de campos e escolha *New Field*. Preencha os campos do editor conforme mostrado a seguir:

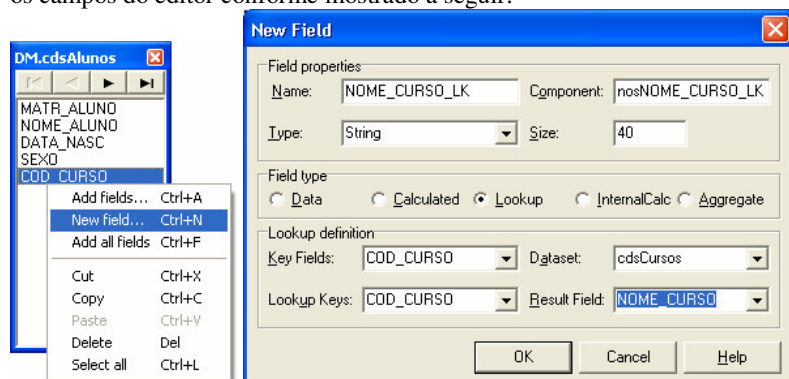


Figura. Configurando um campo Lookup usando ClientDataSet

Após esta operação, ative novamente o *ClientDataSet* de Alunos (*cdsAlunos*), configurando sua propriedade *Active* para *True*. De volta ao editor de campos, arraste o novo campo *Lookup* para o formulário de alunos, ao lado do código do curso, conforme mostrado a seguir:

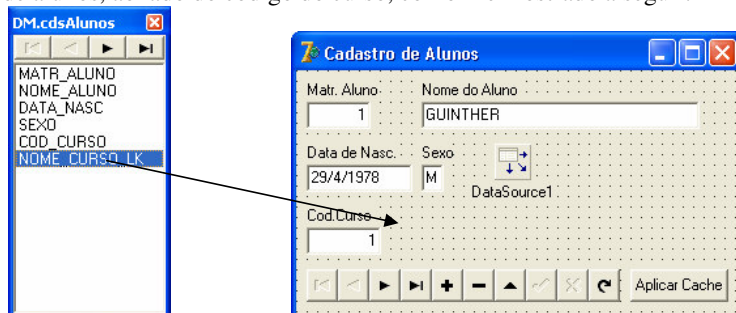


Figura. Criando um DBLookupComboBox.



Figura. Testando o campo Lookup buscando dados de outra tabela (ClientDataSet).

Como funciona a *DBLookupComboBox*? Você seleciona o curso na lista, e o Delphi automaticamente joga o código deste curso no campo *COD_CURSO* da tabela *ALUNOS* como você pode ver na caixa ao lado do *Lookup*. Quando você grava o registro, somente o código do curso é gravado na tabela *ALUNOS*.

O DataSetProvider

Quando criamos a conexão *dbExpress* configuramos também os comandos de busca de dados (*SELECT*) usando a propriedade *SQL* do *SQLQuery*. Mas e quando eu trabalho sobre os dados do formulário, por exemplo a inclusão de um registro, exclusão ou alteração, como são feitas as instruções de *Insert*, *Delete* e *Update*? Lembre-se que trabalhávamos com essas instruções no *IBConsole*. Mas e no *dbExpress*, por que não precisei fornecer esses comandos, apenas o *SELECT*, para trabalhar com os dados da tabela?

Quando chamamos o método *ApplyUpdates* do *ClientDataSet*, tudo o que foi alterado (chamado de **DELTA**) é devolvido ao componente *DataSetProvider*, que se encarrega de atualizar o banco de dados para nós, gerando as instruções de *Update*, *Delete* e *Insert* apropriadas.

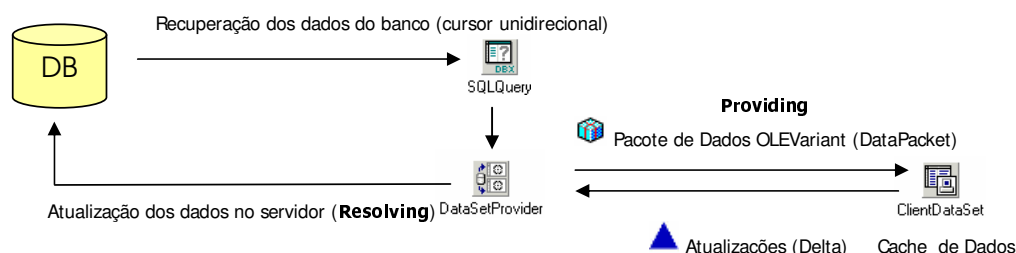


Figura. Esquema de recuperação e atualização de dados na arquitetura dbExpress

1º - O *DataSet* do dbExpress (neste caso uma *SQLQuery*) abre uma conexão com o banco de dados através de um *SQLConnection* e recupera os dados da consulta usando um *Cursor SQL*.

2º - O *DataSetProvider* varre este *DataSet* (usando um loop unidirecional “while not eof”) pegando todos os campos de todos registros e empacotando em uma estrutura chamada **DataPacket**. Este empacotamento é feito por um objeto interno do *DataSetProvider* chamado *DataPacketWriter*. Esse processo é também conhecido como **Providing**.

3º - O pacote de dados é enviado a aplicação Cliente e colocado no Buffer do *ClientDataSet*.

4º - O cursor usado na consulta SQL bem como a conexão são liberados (isso é, eles não mantêm o seu estado).

5º - O cliente altera os dados do buffer do *ClientDataSet*, criando um pacote *Delta* de alterações.

6º - O Delta é enviado ao servidor de aplicação e é recebido pelo *DataSetProvider*, o qual se encarrega de montar uma instrução de atualização (Insert, Update ou Delete). Observe que a atualização é montada sem a devolução dos dados à *SQLQuery*, o qual não pode ser alterado por ser unidirecional. Esse processo é também conhecido como **Resolving** e é feito por um objeto interno ao *DataSetProvider* chamado *Resolver*.

SQL Monitor

Para sabermos como o *DataSetProvider* está gerando as instruções de atualização, e principalmente, “se” está gerando corretamente tais instruções, faça o seguinte:

Coloque no *DataModule* o componente *SQLMonitor* da paleta *dbExpress*, aponte sua propriedade *SQLConnection* para *SQLConnection1*, atribua o valor “C:\SQL_LOG.TXT” para *FileName*, e configure as propriedades *AutoSave* e *Active* para *True*.

Testando: Execute novamente a aplicação, selecione um aluno, altere seu nome, aperte *Gravar* (o “V” da barra de navegação) e depois aplique a cache(delta) pressionando o botão *Aplique Cache*. Como exemplo troquei meu nome de “GUINThER” para “GUINThER PAULI”. Agora abra o arquivo C:\SQL_LOG.TXT, e veja as várias instruções SQL para comunicação entre servidor Interbase e cliente Delphi. Observe a instrução SQL de *update* gerada em consequência da alteração que fizemos na tela do formulário:

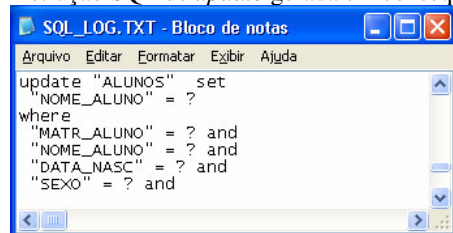


Figura. Usando o SQL Monitor para “depurar” as instruções de atualização do *DataSetProvider*

O sinal ? representa uma instrução **parametrizada**.

Esse código acima funciona, mas não está otimizado. Não precisamos dizer ao Interbase o seguinte: “**Atualize** o Nome do Aluno para GUINThER PAULI **onde** o Código do Curso for 1, o Sexo for M, a Data de Nascimento for 29/04/1978, o Nome for GUINThER e a Matrícula for 1”. Não precisamos especificar

todas as características do Aluno na cláusula **WHERE** para identificá-lo, basta sua matrícula, pois esse é o campo-chave que identifica o aluno. Basta dizer isso ao Interbase: “**Atualize** o Nome do Aluno para GUINThER PAULI onde a Matrícula é 1”. Quando você vai a Locadora o funcionário só pergunta a você seu código (se você já estiver cadastrado). Ele não pergunta pra você seu sexo, sua data de nascimento e seu curso para achar sua ficha, porque apenas pelo seu código na Locadora é possível saber tudo sobre você.

Atenção: algumas aplicações utilizam instruções de *update* como tratamento otimizado para concorrência, mantendo todos os campos na cláusula *where*.

Como não temos acesso direto a esta instrução gerada pelo *Resolver* do *DataSetProvider*, precisamos configurar a instrução de *update* usando *ProviderFlags*.

Provider Flags

O propriedade *ProviderFlags* de um campo *TField* pode especificar como cada campo participará das instruções de atualização geradas pelo *DataSetProvider*, como *Update*, *Insert* e *Delete*.

Siga os passos abaixo para configurar corretamente a atualização de dados da tabela de Alunos:

No *DataModule*, selecione o componente *qryAlunos*. Dê um duplo clique nele e no editor de campos dê um clique direita. Escolha a opção *Add all fields*. Selecione então todos os campos *TFields*, menos *MATR_ALUNO*, e configure a propriedade *ProviderFlags.pfInWhere* para *False*. Isso diz ao *DataSetProvider* que esses campos não devem ser colocados na cláusula *WHERE* das instruções de atualização.

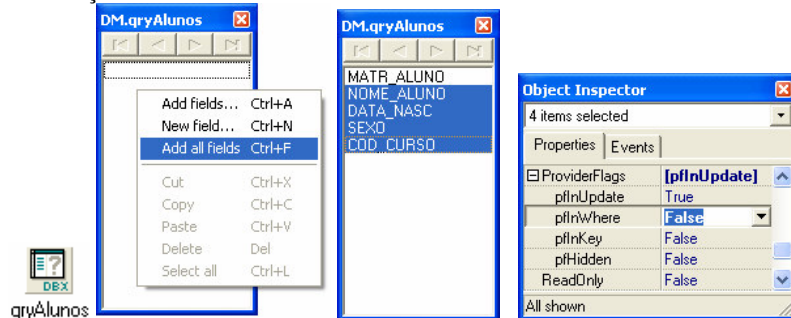


Figura. Configurando corretamente os *ProviderFlags* dos objetos *TFields*

Selecione então apenas o *TField* *MATR_ALUNO* e configure sua propriedade *ProviderFlags.pfInKey* para *True*. Isso diz ao *DataSetProvider* que o *MATR_ALUNO* é chave.

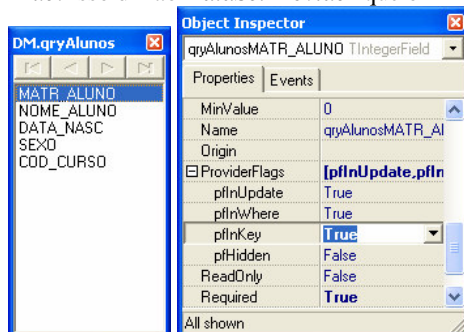


Figura. O campo matrícula é o campo chave

Por último, configure a propriedade *UpdateMode* de *dspAlunos* para *upWhereKeyOnly*. Execute novamente a aplicação e faça uma atualização. Grave os dados e aplique a *Cache (delta)*. Abra o arquivo *C:\SQL_LOG.TXT* e observe a nova instrução de *update*, agora correta:

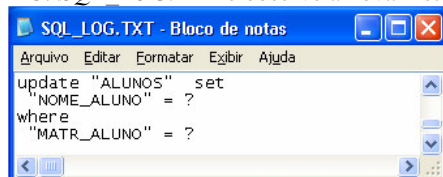


Figura. Nova instrução de Update depois de configurados os ProviderFlags

Atenção: Depois de testar o *SQLMonitor*, configure sua propriedade *Active* para *False*.

Consultas parametrizadas

Observe a instrução *SELECT* do componente *qryAlunos* que recupera os dados da tabela do Interbase:

```
select * from ALUNOS order by NOME_ALUNO
```

Se esse sistema estive rodando em uma escola com muitos alunos, por exemplo 1000 ou 5000 alunos, quando você abrisse esta consulta, **“TODOS OS REGISTROS DA TABELA DE ALUNOS SERIAM TRAZIDOS DO SERVIDOR COLOCADOS NA CACHE DO CLIENTDATASET”**. Isso poderia causar uma enorme lentidão na abertura das tabelas. Pergunto, para que trazer todos os registros da tabela de alunos se nosso usuário vai consultar e alterar apenas três ou quatro registros?

Para isso usamos *Consultas Parametrizadas*. Uma consulta (*query*) parametrizada basicamente é um *SELECT* que tem um parâmetro na cláusula *where*. O valor para este parâmetro é então fornecido em tempo de execução pelo usuário. Em uma aplicação real cliente servidor, você nunca terá uma instrução *select* sem uma cláusula *where*, algo como *select * from tabela*.

Atenção: Consultas parametrizadas são fundamentais em desenvolvimento cliente/servidor e multicamadas.

Para testar esse recurso em nossa aplicação, selecione o componente *qryAlunos* e altere sua propriedade *SQL* para:

```
select
  COD_CURSO,
  DATA_NASC,
  MATR_ALUNO,
  NOME_ALUNO,
  SEXO
from
  ALUNOS
where
  MATR_ALUNO=:PARAM_MATR
```

O *SELECT* acima retorna somente **1** registro, de acordo com uma matrícula que será fornecida pelo usuário. Não há necessidade da cláusula *order*, pois não é há porque ordenar um conjunto de dados que só tenha um registro. Depois de alterar o *SQL* pressione *OK*.

Vá até a propriedade *Params*. Selecione o único parâmetro indicado no editor, e configure a propriedade *DataType* para *ftInteger*. Observe que *ParamType* é do tipo *ptInput* (entrada).

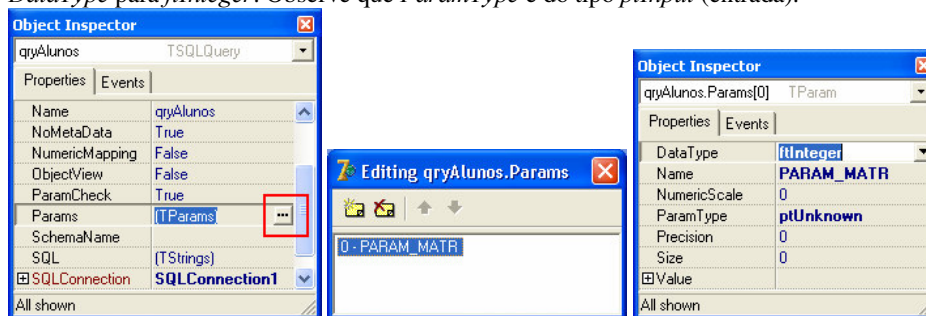


Figura. Configurando a entrada de parâmetros na instrução *SQL*

Depois de configurado o parâmetro, desative o *ClientDataset* de Alunos (*cdsAlunos*), configurando sua propriedade *Active* para *False*. Ele só será aberto depois que definirmos o parâmetro. Dê um clique de direita sobre o *cdsAlunos* e escolha a opção *Fetch Params*. Isso busca os parâmetros definidos em *qryAlunos*.

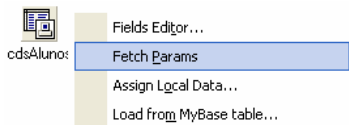


Figura. Trazendo os parâmetros da SQLQuery para o ClientDataSet

Coloque no formulário uma Label com Caption de “Abrir Ficha do Aluno:”, e um Edit, sem texto. Depois coloque um botão, e no seu evento OnClick digite:

```
if Edit1.Text='' then
begin
  exit;
dm.cdsAlunos.close;
dm.cdsAlunos.Params[0].AsString:=Edit1.Text;
dm.cdsAlunos.Open;
if dm.cdsAlunos.IsEmpty then
  MessageDlg('Não há nenhum aluno com a Matrícula Informada!', mtWarning, [mbOK], 0);
```

Abaixo você pode ver o a abertura da consulta parametrizada:

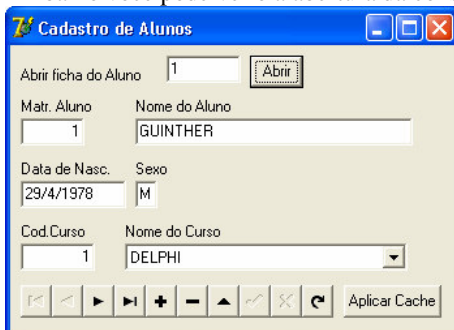


Figura. Abrindo a ficha de um Aluno usando uma consulta parametrizada

Por que SQLQuery + DataSetProvider + ClientDataSet ?

Como vimos anteriormente, para acessar a tabela de ALUNOS precisamos colocar 3 componentes no *DataModule* (*SQLQuery*+*DataSetProvider*+*ClientDataSet*). Usar esses componentes nos dão um controle enorme sobre o recuperação, tratamento e atualização dos dados. Um exemplo disso é termos os *TFields* na *SQLQuery* que determinam as instruções de *update* e os *TFields* no *ClientDataSet* que fornecem o tratamento cliente, como máscaras e validações simples.

Se você sempre utilizar esses três componentes, poderá facilmente migrar seu aplicativo duas camadas para três camadas quando for necessário, ou ainda alternar entre diferentes tecnologias de acesso a dados, como *IBX*, *dbExpress*, *BDE* ou *ADO*, sem perder seu código ou ter que refazer todo o seu *DataModule*.

Para fazer a migração, simplesmente retire do *DataModule* os componentes que fazem acesso ao banco, e coloque-os em um *Remote DataModule* ou *Transactional DataModule* (observe a figura):

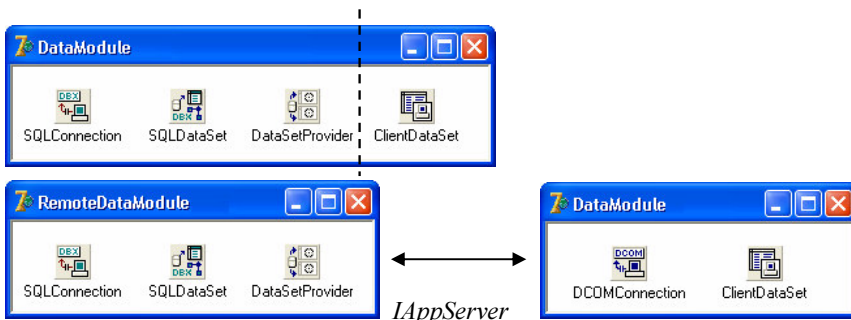


Figura. Migração de uma aplicação duas-camadas para três-camadas

SimpleDataSet e SQLClientDataSet

É claro que a Borland simplificou um pouco as coisas no *dbExpress* para quem não vai migrar para 3 camadas. Existe um super componente “stand-alone” que faz o papel de todos os personagens acima.

No Delphi 6, este componente se chamava *SQLClientDataSet*, que possui **internamente** o tradicional trio de componentes, como mostra o esquema abaixo:

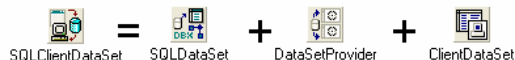


Figura. *SQLClientDataSet*

Usar o *SQLClientDataSet* ao invés do trio não lhe dará um total controle sobre a situação. Por exemplo, ele não publica as propriedades do *SQLDataSet*, logo não podemos configurar as *ProviderFlags*, comprometendo as instruções de atualização. Por isso, ele é uma ótima solução em consultas que **precisam de cache mas não serão atualizadas no banco**.

O Delphi 7 não traz mais o *SQLClientDataSet*, introduzindo um componente semelhante e com a mesma função. Este novo componente é o *SimpleDataSet*, que corrige alguns problemas do *SQLClientDataSet* e implementa algumas melhorias.



Figura. *SimpleDataSet*

Uma das melhorias do *SimpleDataSet* é ter um componente interno de conexão, e o mais importante, tempos acesso as propriedades e *TFields* do *SQLDataSet* interno.

Tenha em mente que usando qualquer umas das alternativas acima, você estará fortemente dependente da arquitetura 2-tier e do *dbExpress*. Lembre-se ainda que o *SimpleDataSet* não existe no Kylix ainda (até a vs. 3). Lembre também que o *SQLClientDataSet* não acompanha mais o Delphi 7, mas pode ser instalado separadamente, caso você precise de compatibilidade com o Delphi 6 / Kylix.

Consultas parametrizadas e Procura

Voltando ao nosso exemplo, pense na consulta parametrizada pela matrícula do aluno. Mas e se não soubéssemos a número da matrícula do Aluno, como abríríamos sua Ficha?

Para resolver este problema, precisamos fazer uma consulta auxiliar, que pelas iniciais do nome do aluno possa retornar sua matrícula. Então pegamos a matrícula dessa pesquisa auxiliar e jogamos no parâmetro do *select* principal.

Para criar essa consulta auxiliar, precisaríamos colocar outro componente *SQLQuery* no *DataModule*, outro *DataSetProvider* e outro *ClientDataSet*. Para facilitar podemos utilizar o componente *SimpleDataSet* da paleta *dbExpress* (ou *SQLClientDataSet* se for D6).

Coloque então no *DataModule* um componente *SQLQuery* e configure suas propriedades como mostrado a seguir:

qryLocalizarAluno	dspLocalizarAluno	cdsLocalizarAluno
Name = qryLocalizarAluno	Name = dspLocalizarAluno	Name = cdsLocalizarAluno
Connection = SQLConnection1	DataSet = qryLocalizarAluno	ProviderName = dspLocalizarAluno
SQL= select A.MATR_ALUNO, A.NOME_ALUNO, C.NOME_CURSO from ALUNOS A, CURSOS C where A.COD_CURSO=C.COD_CURSO order by A.NOME_ALUNO		

Dê dois clique no *ClientDataSet*, dê em clique de direita no editor de campos e escolha *Add All Fiels* para adicionar os campos *TFields*. Configure a propriedade *DisplayLabel* dos *TFields* para “Matr.Aluno”, “Nome do Aluno” e “Curso”.

Observe que trazemos somente os dados fundamentais para a identificação do aluno, como seu nome, matrícula e curso. Não ative esse *ClientDataSet* pois ele será aberto em runtime.

Crie um novo formulário de busca, clicando em *File|New|Form*. Salve o novo formulário como “uFrmLocalizarAluno” e configure sua propriedade *Name* para “FrmLocalizarAluno”. Pressione ALT+F11 e escolha a unit *uDM*, pois precisamos referenciar componentes do *DataModule*. Coloque nesse formulário um componente *DataSource* da paleta *DataAccess*. Aponte a propriedade *Dataset* do *DataSource* para *DM.cdsLocalizarAluno*. Coloque no formulário uma *StatusBar* (paleta *Win32*), configurando sua propriedade *SimplePanel* para *True*. Coloque um componente *DBGrid* da paleta *DataControls*, aponte sua propriedade *DataSource* para *DataSource1* e configure sua propriedade *Align* para *alBottom*. Configure sua propriedade *Options.dgRowSelect* para *True*. Dê dois cliques no *DGrid*, e no editor de colunas clique de direita e escolha a opção *Add All Columns*. Configure a aparência das colunas usando as propriedades *TitleFont* de cada coluna e *FixedColor* do *DBGrid*. Finalmente coloque um botão no formulário com o *Caption* de “Localizar”. Observe o layout da busca na figura a seguir:

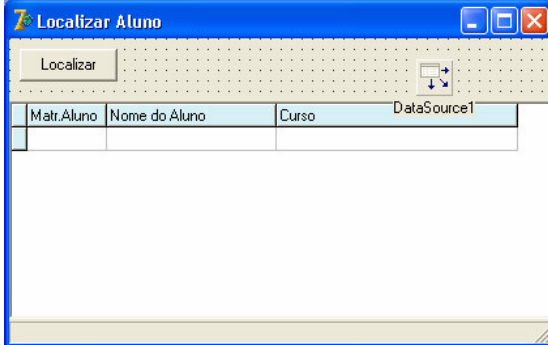


Figura. Construindo um formulário de procura

No evento *OnClick* do botão digite o seguinte:

```
DM.cdsLocalizarAluno.Open; //abre a consulta
```

Selecione o *DBGrid* e no seu evento *OnDbClick* digite:

```
close;
```

Volte ao formulário principal, coloque um novo botão ao lado do botão *Abrir*, dê o *Caption* de “Localizar...” e no seu evento *OnClick* digite:

```
FrmLocalizarAluno.ShowModal; // mostra o formulário
DM.cdsAlunos.close;
(* o código abaixo atribui ao parametro do ClientDataset
o valor da matrícula escolhido na busca *)
DM.cdsAlunos.Params[0].AsInteger:=
DM.cdsLocalizarAluno.FieldByName('MATR_ALUNO').AsInteger;
DM.cdsAlunos.Open; // abre a ficha do aluno
DM.cdsLocalizarAluno.close; // fecha e libera a busca
```

Execute a aplicação e teste a busca.

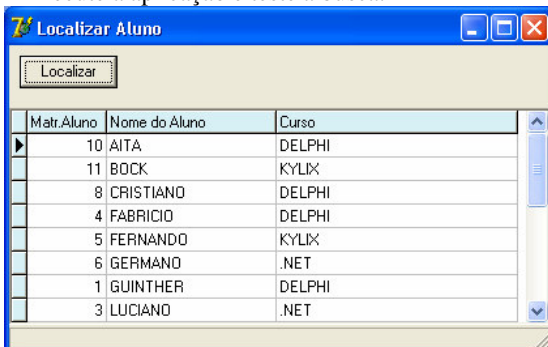


Figura. Campo de uma consulta SQL é usado como parâmetro de outra consulta

Você pode otimizar ainda mais a busca pedindo para que o usuário forneça apenas as iniciais do nome que deseja localizar. Para isso, altere o *SQL* de *qryLocalizarAluno* para:

```
select
  A.MATR_ALUNO,
  A.NOME_ALUNO,
  C.NOME_CURSO
from
  ALUNOS A,
  CURSOS C
where
  A.COD_CURSO=C.COD_CURSO and
  UPPER(A.NOME_ALUNO) like :PARAM_NOME
order by
  A.NOME_ALUNO
```

Vá até a sua propriedade *Params* e selecione o único parâmetro da lista. Configure sua propriedade *DataType* para *ftString*. Observe a figura:

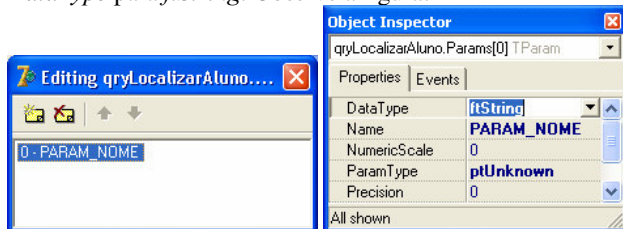


Figura. Configurando o parâmetro de entrada (Nome) de *cdsLocalizarAluno*

Dê um clique de direita em *cdsLocalizarAlunos* e escolha a opção *Fetch Params*. Volte ao formulário de busca (*FrmLocalizarAluno*) e coloque um *Edit* ao lado do botão, retirando seu texto. Agora altere o código do evento *OnClick* do botão de procura para:

```
DM.cdsLocalizarAluno.close; //fecha se estiver aberta
DM.cdsLocalizarAluno.params[0].asString:=UpperCase(Edit1.Text+'%'); //iniciais mais coringa
DM.cdsLocalizarAluno.Open; //abre query de busca
StatusBar1.SimpleText:=format('Foram encontrados %d registros iniciando com "%s"',
  [DM.cdsLocalizarAluno.recordcount,edit1.text]);
```

Atenção: Caso você esteja usando *SQLClientDataSet* retire a referência “dataset” (em negrito).

Teste a busca, digitando as iniciais do aluno que você quer encontrar. Clicando sobre o registro, a ficha do aluno será então aberta.



Figura. Busca parametrizada com LIKE

Joins ao invés de Lookups

Vimos que nossa tabela de CURSOS possui poucos registros. Mas e se esta tabela tivesse 1000 registros (cursos), ao abrir uma ficha de um aluno, seriam trazidos todos os 1000 cursos para o cliente, mesmo que na tela só apareça o nome do curso do aluno. Isso porque usamos um campo *Lookup*, que pode mostrar em uma caixa suspensa todos os cursos cadastrados. Em sistemas reais cliente/servidor, um *Lookup* não deve ser

utilizado quando a tabela alvo possui muitos registros, pois isso pode ocasionar uma busca desnecessária de dados.

Para resolver o problema, altere a instrução *select* da propriedade SQL de *qryAlunos* para que já traga o nome do curso já a partir do servidor, para que a operação não seja feita no cliente:

```
select
  A.MATR_ALUNO,
  A.NOME_ALUNO,
  A.COD_CURSO,
  A.DATA_NASC,
  A.SEXO,
  C.NOME_CURSO
from
  ALUNOS A,
  CURSOS C
where
  A.COD_CURSO=C.COD_CURSO and
  MATR_ALUNO=:PARAM_MATR
```

Dê em duplo clique em *qryAlunos*, depois um clique de direita, e escolha *Add Fields*. Adicione então o *TField* *NOME_CURSO*. Altere as propriedades *ProviderFlags.pfInUpdate* e *pfInWhere* para *False*. Esse campo será usando somente para visualização e não deve ser usado para instruções de *update*.

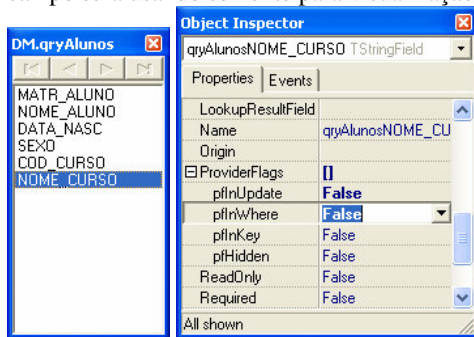


Figura. Campo NOME_CURSO não será atualizado

Dê um duplo clique em *cdsAlunos*, e no editor de campos **apague** o campo *LK_NOME_CURSO* (que é um *Lookup*). Clique de direita sobre o editor e escolha *Add Fields*. Adicione agora o novo campo *NOME_CURSO*, que é resultado de um JOIN feito no servidor, e não um *Lookup*.

Volte ao formulário de alunos e apague o componente *DBLookupComboBox*. Coloque então um componente *DBText*, e aponte sua propriedade *DataSource* para *DataSource1* e *DataField* para *NOME_CURSO*.

Se você executar a aplicação verá que o nome do curso já está unido pelo JOIN, e não usando *Lookup*. Lembre-se que *Lookup* poder ser usado em tabelas pequenas, mas em sistemas cliente/servidor com tabelas grandes use a metodologia aplicada neste exemplo.

Nosso *ClientDataSet*, na verdade, enxerga o nome do curso como se fosse um campo da própria tabela de alunos.

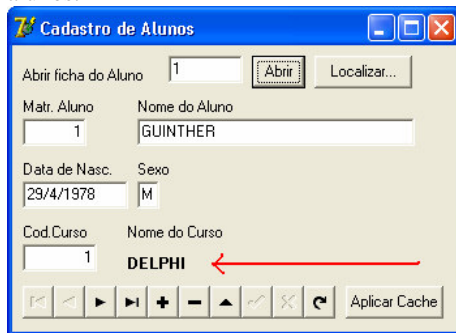


Figura. O campo NOME_CURSO agora é trazido através de um JOIN feito no servidor, e não Lookup cliente

Exercício: Construir uma busca auxiliar (semelhante à feita para buscar um aluno pelo nome) que permita alterar o campo COD_CURSO.

Views

Uma *view* basicamente é uma instrução de *select* que fica armazenada no servidor e que do ponto de vista cliente se comporta como uma TABELA. Uma *view* fornece uma visão para determinados dados de tabelas, não sendo uma entidade física, e sim uma representação lógica de informações relacionadas, ou que obedecem determinada condição, etc.. Você pode, por exemplo, usar uma *view* para limitar um conjunto de campos para uma tabela, de forma que a estação cliente terá acesso a apenas uma "visão" parcial dos dados, e não o conjunto inteiro.

Como exemplo, vá até o *IBConsole* e no *Interactive SQL* digite o seguinte:

```
create view ALUNOS_CURSOS as
select
  A.NOME_ALUNO,
  C.NOME_CURSO
from
  ALUNOS A
inner join
  CURSOS C
on
  C.COD_CURSO = A.COD_CURSO
```

Esta instrução ficará armazenada no servidor. Agora, a partir do cliente ou do próprio *IBConsole*, você pode digitar o seguinte:

```
select * from ALUNOS_CURSOS
```

Note que um *join* já está subentendido no *select* acima.

Triggers e Exceptions

Um *trigger* é um evento que é disparado sempre que alguma operação acontece sobre uma tabela. Por exemplo, podemos criar um *trigger* para monitorar novos registros inseridos em uma tabela e antes de serem gravados fazer a sua consistência. Um *trigger* também pode ser utilizado para excluir ou atualizar registros de tabelas relacionadas à que está sendo atualizada.

No *IBConsole*, crie agora uma *exception* que conterà a descrição do erro:

```
create exception ERROR_SEXO_INVALIDO
'Sexo deve ser "M" ou "F"'
```

Execute esta instrução para criar a *exception*. Depois digite o comando abaixo para criar uma *trigger*.

```
set term # ;
create trigger INSERT_SEXO_ALUNO for ALUNOS
before insert
as
begin
  if (not (NEW.SEXO in ('M','F'))) then
    exception ERROR_SEXO_INVALIDO;
  end #
set term ; #
```

Tente agora inserir um usuário com o sexo inválido:

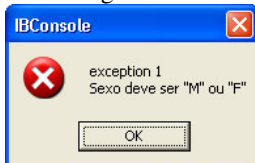


Figura. Usando Exceptions e Trigger para validação Server-Side

Atenção: a validação anterior normalmente deve ser client-side. Apenas a usamos para fins didáticos.

Para validar se um nome já está inserido no banco de dados, primeiro crie esta *exception*:

```
create exception ERROR_NOME_ALUNO_DUPLICADO
'Este nome já está cadastrado no banco de dados!'
```

Execute o comando acima. Agora crie o seguinte *trigger*:

```
set term # ;
create trigger INSERT_NOME_ALUNO for ALUNOS
before insert
as
begin
  if (exists
      (select NOME_ALUNO from ALUNOS
       where NOME_ALUNO=NEW.NOME_ALUNO)) then
    exception ERROR_NOME_ALUNO_DUPLICADO;
  end #
set term ; #
```

Teste a validação tentando incluir um nome repetido no banco de dados:

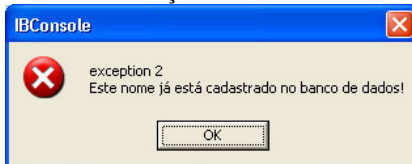


Figura. Usando Triggers e Exceptions para validar nomes repetidos.

Para testar essa validação a partir do Delphi, digite o seguinte comando no evento *OnReconcileError* de *cdsAlunos*:

```
raise Exception.Create(E.Message);
```

Transações em dbExpress

Uma transação garante a atomicidade e consistência de uma operação sobre um conjunto de dados. Como exemplo, coloque um botão com o *Caption* de “Transação” no formulário e digite no seu evento *OnClick*.

```
procedure TFrmAlunos.Button2Click(Sender: TObject);
var
  TD : TTransactionDesc; //declare DBXpress em uses
begin
  TD.TransactionID:=1; //um n° qualquer que identifique a transação
  TD.IsolationLevel:=xlREADCOMMITTED; //nível de isolamento da transação
  dm.SQLConnection1.StartTransaction(TD); //inicia a transação
  DM.SQLConnection1.ExecuteDirect('delete from ALUNOS');
  if MessageDlg('Commit?', mtconfirmation, [mbok, mbcancel], 0) = mrOk then
    DM.SQLConnection1.Commit(TD) //confirma
  else
    DM.SQLConnection1.Rollback(TD); //cancela
end;
```

Atenção: O bloco anterior deve ser protegido com *try finally*, como veremos no Módulo 3.

Atenção: Quando damos usamos o método *ApplyUpdates* para aplicar a cache de um *ClientDataSet*, automaticamente é criada uma transação se nenhuma estiver ativa.

Stored Procedures

Stored Procedures são procedimentos que ficam armazenadas no servidor SQL. Como exemplo, vamos fazer uma *Stored Procedure* que retorna quantos alunos estão matriculados em um determinado curso.

Abra o *IBCConsole* e no *Interactive SQL* digite:

```
SET TERM !! ;
create procedure NUM_ALUNOS_CURSO (COD_CURSO integer)
```

```

returns (NUM_ALUNOS integer)
as
begin
  select
    count (MATR_ALUNO)
  from
    ALUNOS
  where
    COD_CURSO=:COD_CURSO
  into :NUM_ALUNOS;
end !!
SET TERM ; !!

```

Aperte CTRL+E para executar a instrução. No Delphi, coloque um componente *SQLStoredProc* (paleta *dbExpress*) no *DataModule*. Configure suas propriedades *SQLConnection* e *StoredProcName*.

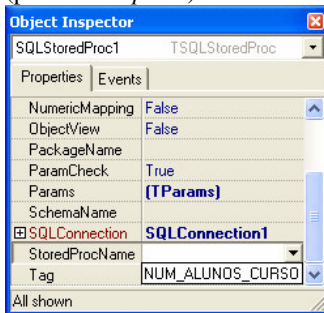


Figura. Acessando um Stored Procedure do Servidor

Agora abra sua propriedade *Params*. Observe que o Delphi já cria os dois parâmetros definidos no Interbase (COD_CURSO é um parâmetro de **INPUT** e NUM_ALUNOS é um parâmetro de **OUTPUT**).

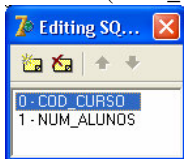


Figura. Parâmetros do SQLStoredProc

Coloque no formulário um *Edit*, um botão e uma *Label*, um ao lado do outro. No evento *OnClick* do botão digite:

```

DM.SQLStoredProc1.Params.ParamByName('COD_CURSO').AsInteger:=StrToInt(Edit2.Text);
DM.SQLStoredProc1.ExecProc;
Label8.Caption:=format('Há %d alunos no curso %s',
  [dm.SQLStoredProc1.Params[1].asinteger,Edit2.Text]);

```

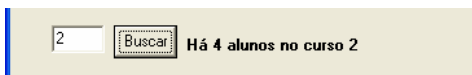


Figura. Executando uma StoredProc no servidor usando parâmetros de Input e Output

Dica: Você também pode usar o componente *SQLDataSet* para acessar uma *Stored Procedure*.






4. Relatórios com QuickReport usando SQL

Instalando o QuickReport no Delphi 7

Clique em *Components*|*Install Packages*>*Add* e selecione o arquivo *\$(Delphi)/bin/dclqrr70.bpl* (marque a opção default)

Criando a consulta

Coloque no DM os componentes abaixo:

		
<i>Name = qryRelatorioAluno</i>	<i>Name = dspRelatorioAluno</i>	<i>Name = cdsRelatorioAluno</i>
<i>Connection = SQLConnection1</i>	<i>DataSet = qryRelatorioAluno</i>	<i>ProviderName = dspRelatorioAluno</i>
<i>SQL=</i> <pre>select * from ALUNOS_CURSOS</pre>		

Observe que o código acima faz *select* em uma *view*. Ative *cdsRelatorioAluno* (apenas para testar relatório em design-time, depois feche-o; normalmente ele deve ser aberto em runtime).

Relatório Simples

Clique em *File|New|Other|Report*. Salve o relatório como “uQRAlunos” e configure sua propriedade *Name* para *QRAlunos*. Aperte ALT+F11 e escolha *uDM* para que o relatório possa reconhecer os componentes do *DataModule*.

Agora um passo **importante**: configure a propriedade *DataSet* do relatório para que possa apontar para *cdsRelatorioAlunos* do relatório que está no *DataModule* (se não aparecer nada na listagem no *Object Inspector* ou seu relatório sair em branco (ou mostrando apenas um registro) é porque você não efetuou esse procedimento).

Dê um duplo clique no relatório e marque as opções como mostra a figura a seguir.

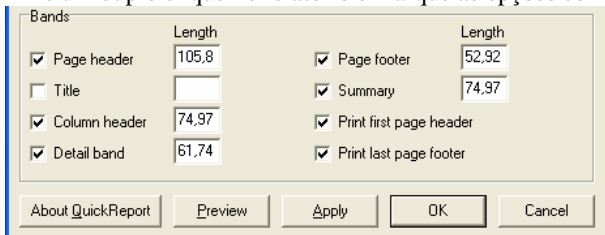


Figura. Escolhendo quais “bandas” farão parte do Relatório

Agora você deve usar os componentes da paleta *QReport* para montar o relatório.

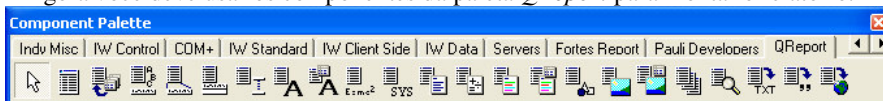


Figura. Paleta QReport

Adicione os componentes como mostra a figura a seguir:

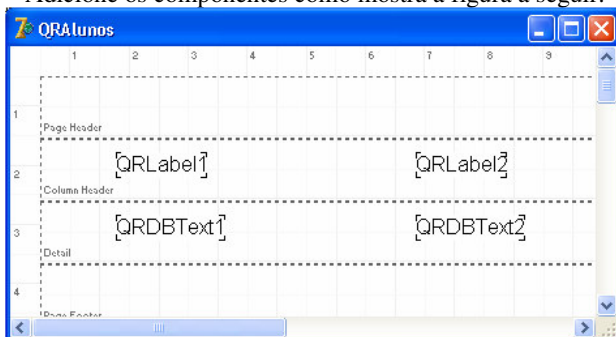


Figura. Adicionando QLabels e QDBTexts ao relatório.

Configure o *Caption* dos *QLabels* para “Nome do Aluno” e “Nome do Curso”. Configure a propriedade *DataSet* dos *QDBTexts* para *DM.cdsRelatorioAlunos* e aponte a propriedade *DataField* para *NOME_ALUNO* e *NOME_CURSO* respectivamente.

Você pode dar um título para o relatório colocando uma *QRLabel* na banda *Page Header*. Coloque também um componente *QRSysData* na banda *PageFooter*, defina sua propriedade *Data* como *qrsPageNumber* e configure seu *Text* para "Página ". Utilize o mesmo componente para colocar um contador de registros no formulário ou fazer uma numeração dos registros (1,2,3,4,...). Ele ainda pode ser utilizado para exibir a Data/Hora do sistema. Coloque também um *QRImage* no *PageHeader* e escolha uma figura para o relatório.

Dê um clique de direita no relatório e escolha *Preview* para visualizar a impressão:

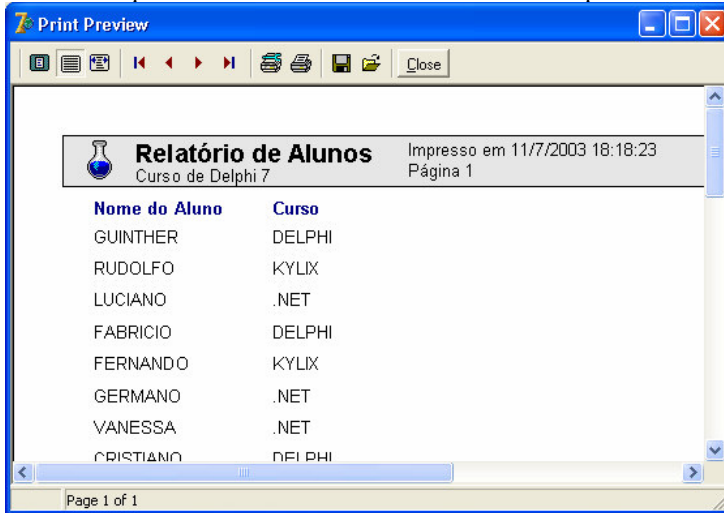


Figura. Visualizando o relatório

Sumários

Coloque um componente *QRExpr* na banda *Summary*. Na sua propriedade *Expression* digite *COUNT*. Coloque um *QRLabel* do seu lado esquerdo e escreva no seu *Caption* "Nº de Alunos". Veja o resultado na figura a seguir:

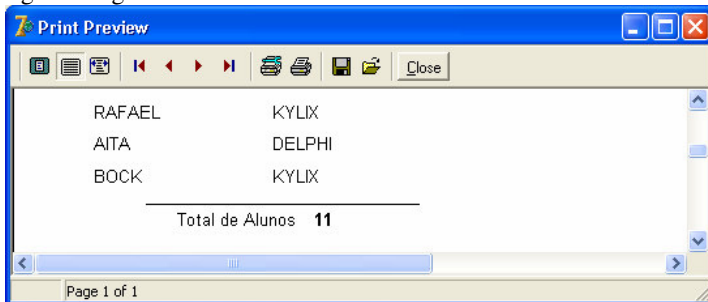


Figura. Usando QRExpression

Relatórios agrupados

Altere a propriedade *SQL* de *qryRelatorioAlunos* do relatório para o código a seguir.

```
select *
from
  ALUNOS_CURSOS
order by
  NOME_CURSO
```

Reabra o *ClientDataSet*. Coloque no *QuickReport* um *QRGroup* da paleta *QReport*. Retire o *QRDBText* que mostra o nome do curso da banda *Detail* e coloque-o na banda *QRGroup*. Remova também a banda *ColumnHeader*. Vá até a propriedade *Expression* da banda *QRGroup* e digite o seguinte:

```
cdsRelatorioAluno.NOME_CURSO
```

Você já pode dar um *preview* no relatório para visualizar os dados.

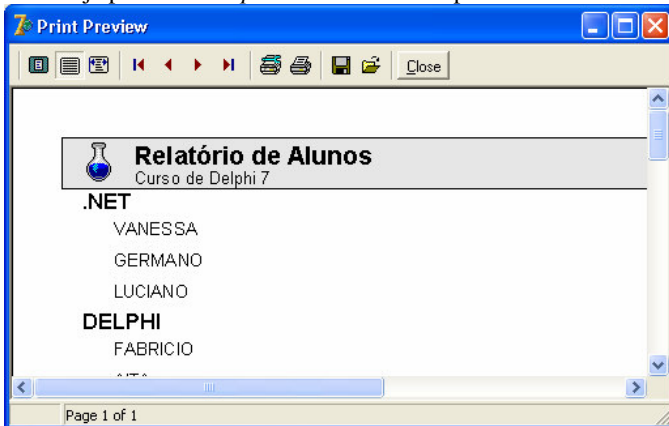


Figura. Relatório Agrupado

Agora vamos sumarizar os valores dentro dos grupos. Coloque no relatório um componente *QRBand*. Configure seu *Name* para “SumarioGrupo” e *BandType* para *rbGroupFooter*. Coloque dentro desta banda um componente *QRExpr*, configurando sua propriedade *Expression* para *COUNT* e *ResetAfterPrint* para *True*. Coloque um *QRLabel* do seu lado esquerdo com o *Caption* de “Total do Curso”. E por último selecione a banda *QRGroup* e aponte sua propriedade *FooterBand* para *SumarioGrupo*. Observe o resultado:

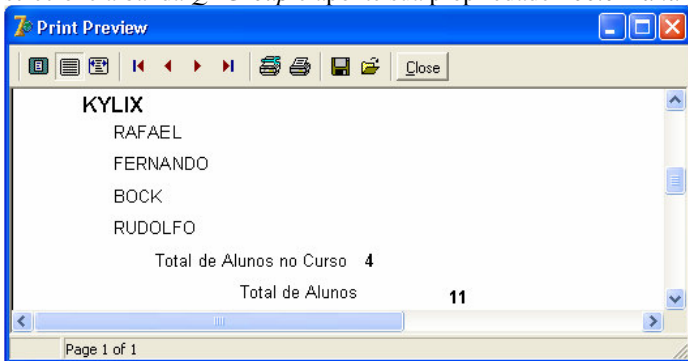


Figura. Relatório com totalizadores por grupo

Depois que você testar o relatório, desative o *SimpleDataset* do relatório que está no *DataModule*. Coloque no formulário um botão com o *Caption* de “Relatório...” e no seu evento *OnClick* digite:

```
dm.cdsRelatorioAluno.Open;
QRAlunos.Preview;
dm.cdsRelatorioAluno.close;
```

5. Gráficos com DBChat usando SQL

Coloque no *DM* um componente *SQLQuery* suas propriedades como abaixo:

<i>Name</i> = <i>qryGraficoAlunosCurso</i>
<i>Connection</i> = <i>SQLConnection1</i>
<i>SQL</i> = <pre>select C.NOME_CURSO, count(A.MATR_ALUNO) from ALUNOS A inner join CURSOS C on C.COD_CURSO = A.COD_CURSO group by C.NOME_CURSO</pre>

Ative o *qryGraficoAlunosCurso* (apenas para teste do gráfico em design-time, depois feche-o). Crie um novo formulário e salve-o com o nome de “uFrmGraficoAlunosCurso”, e dê a ele o nome de “FrmGraficoAlunosCurso”. Aperte ALT+F11 e selecione *uDM* para que ele possa reconhecer o *qryGraficoAlunosCurso*.

Coloque no form um componente *DBChart* da paleta *DataControls*. Dê um duplo clique no *DBChart*. Clique em *Add*. Escolha o tipo *Pie*. Vá até a guia *Series* e depois *DataSource*. Agora configure as opções como mostrado a seguir:

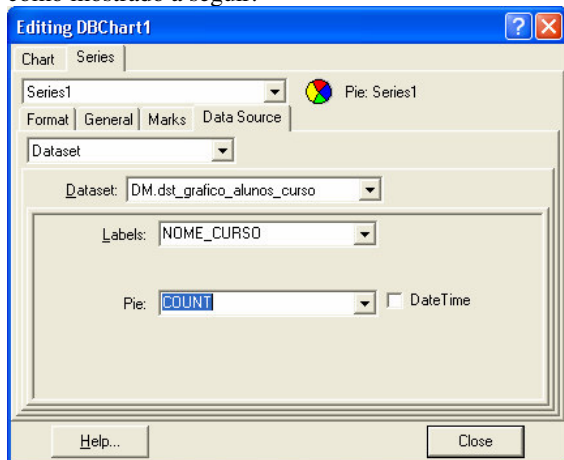


Figura. Configurando parâmetros do DBChart

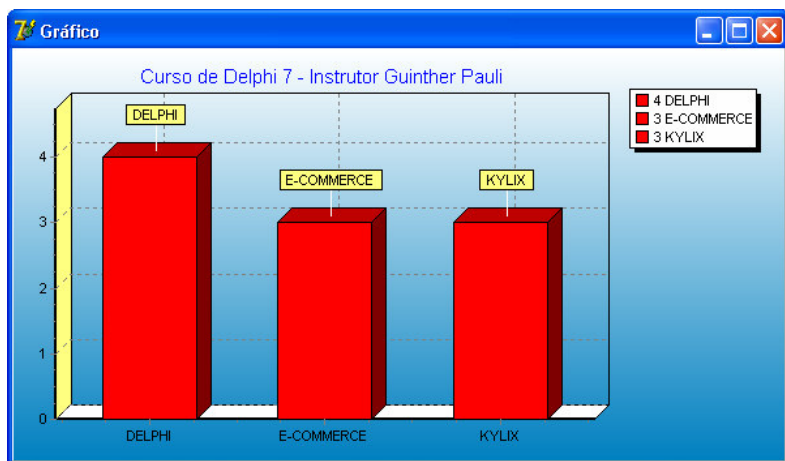


Figura. Gráfico de Total de Alunos inscritos por Curso

Após testado o *DBChart*, feche *qryGraficoAlunosCurso*. Agora coloque um botão no formulário de alunos com o *Caption* de “Gráfico”, e escreva o seguinte no seu código *OnClick*:

```
DM.QryGraficoAlunoCurso.Open;
FrmGraficoAlunoCurso.showmodal;
DM.QryGraficoAlunoCurso.Close;
```

6. XML com Delphi 7

Sintaxe Básica da XML

Existem alguns elementos técnicos da XML que vale a pena conhecer antes de sua utilização dentro do Delphi. Aqui está um breve resumo dos principais elementos da sintaxe da XML:

- Espaço em branco é ignorado;

- Você pode adicionar comentários dentro de marcas `<!-- e -->`;
- Existem diretivas e instruções de processamento, incluído dentro das marcas `<? e ?>`;
- As marcas são incluídas entre os sinais de menor e maior ("`<`" e "`>`") e diferem entre maiúsculas e minúsculas;
- Para cada marca de abertura, você deve ter uma marca de fechamento correspondente, representada por um caractere de barra inicial, como em:

```
<no> valor </no>
```

- As marcas não devem se sobrepor e devem estar corretamente aninhadas, como a seguir:

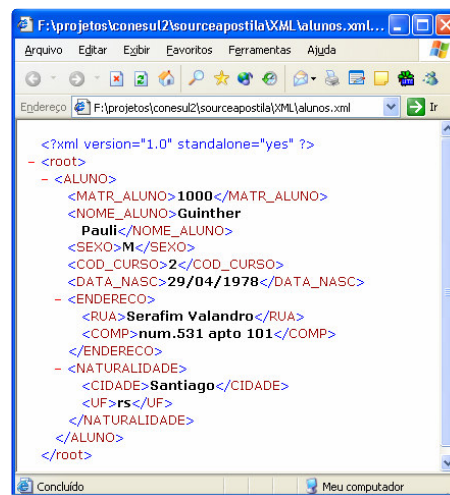
```
<no1> xx <no2> yy </no2> </no1> // CERTO
<no1> xx <no2> yy </no1> </no2> // ERRADO
```

Documentos XML

Abra o Bloco de Notas e digite o seguinte código XML a seguir. Salve este arquivo como *alunos.xml*. Arraste o arquivo para o dentro do *Internet Explorer*.

```
<?xml version="1.0" standalone="yes" ?>
<root>
  <ALUNO>
    <MATR_ALUNO>1000</MATR_ALUNO>
    <NOME_ALUNO>Guinther Pauli</NOME_ALUNO>
    <SEXO>M</SEXO>
    <COD_CURSO>2</COD_CURSO>
    <DATA_NASC>29/04/1978</DATA_NASC>
    <ENDereco>
      <RUA>Serafim Valandro</RUA>
      <COMP>num.531 apto 101</COMP>
    </ENDereco>
    <NATURALIDADE>
      <CIDADE>Santiago</CIDADE>
      <UF>rs</UF>
    </NATURALIDADE>
  </ALUNO>
</root>
```

Figura. Documento XML



A estrutura deste documento XML pode ser representada na tabela abaixo:

ALUNOS								
MATR_ALUNO	NOME_ALUNO	SEXO	COD_CURSO	DATA_NASC	ENDereco		NATURALIDADE	
1000	Guinther Pauli	M	2	29/04/1978	RUA	COMP	CIDADE	UF
					Serafim Valandro	num531 apto101	Santiago	RS

Agora insira dois registros no arquivo XML (antes da tag `</root>`, abra uma nova tag `</pessoa>` e insira novos valores dentro dos elementos (*matr_aluno*, *nome_aluno*, etc).

ClientDataSet e XML DataPacket

Como vimos no módulo 1 podemos trabalhar com documentos XML Data Packet usando o *ClientDataSet*. Em uma nova aplicação Dlephi coloque um *ClientDataSet* no formulário. Abra sua propriedade *FileName* e selecione o arquivo que criamos anteriormente (*Alunos.xml*).

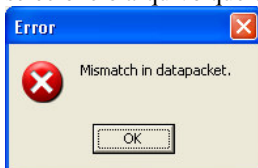


Figura. Erro ao tentar abrir nosso Documento XML no ClientDataSet

Por que isso acontece? Porque nosso documento XML não é um *DataPacket*.

XML Mapper

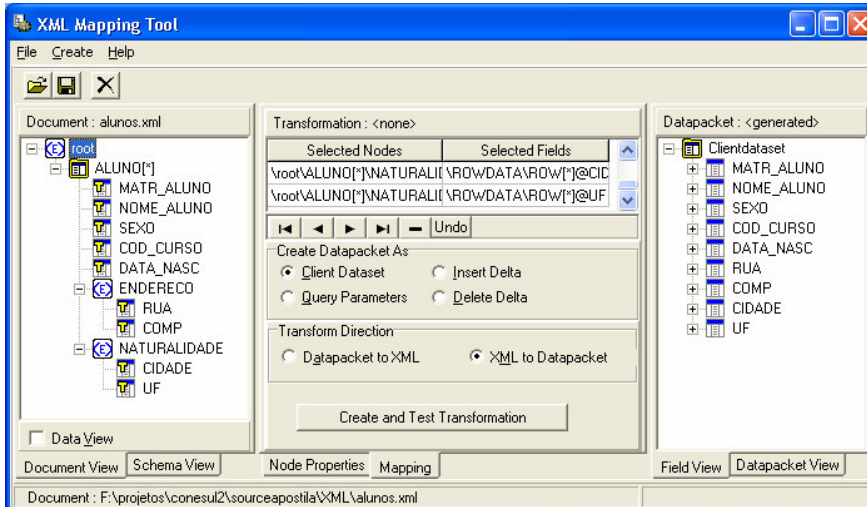


Figura. Tela principal do XML Mapper

Abra o XML Mapper (*Iniciar/Programas/Borland Delphi 7/XML Mapper*). Clique em *File/Open*. Escolha então o arquivo XML criado anteriormente. Selecione o campo *MATR_ALUNO* e clique na guia *Node Properties* e altere seu *Data Type* para *integer*.

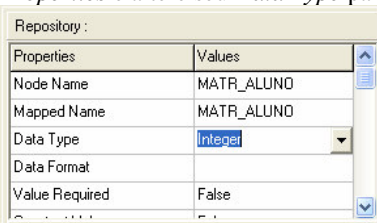


Figura. Configurando propriedades de um nó

Da mesma forma, escolha o campo *NOME_ALUNO* e defina *Max Length* para 60. Defina *COD_CURSO* como *Data Type integer*. Para *DATA_NASC* defina *Date Type* como *date* e *Data Format* como *dd/mm/yyyy*. Configure *RUA* com *Max Length* como "40", *COMP* como "20", "CIDADE" como 40 e "UF" como 2.

Dê um clique de direita sobre o nó *root* no painel da esquerda e escolha a opção *Select All Children*. Verifique se a direção da transformação está como *XML to DataPacket*.

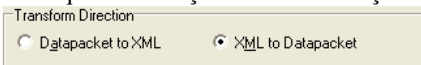


Figura. Direção da transformação

Clique no menu *Create/DataPacket from XML*. Isso criará uma *DataPacket* a partir do *XML Document*.

Agora clique neste botão

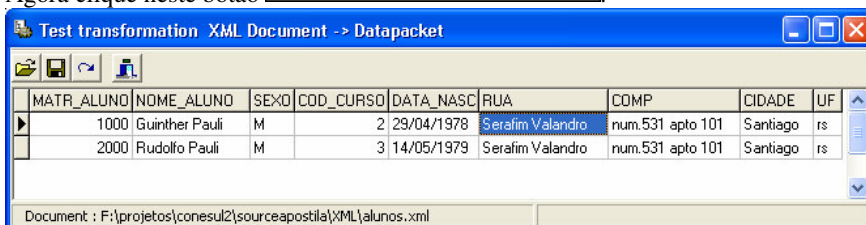


Figura. Configurando propriedades de um nó

Feche o *preview* e na janela principal do *XML Mapper* clique em *File|Save|Transformation*. Salve o arquivo gerado como *xml_to_datapacket.xtr*. De volta a janela principal do *XML Mapper*, troque a direção da transformação para *DataPacket to XML*.

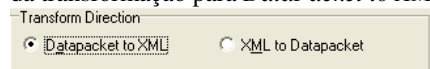



Figura. Direção da transformação

Agora clique neste botão 

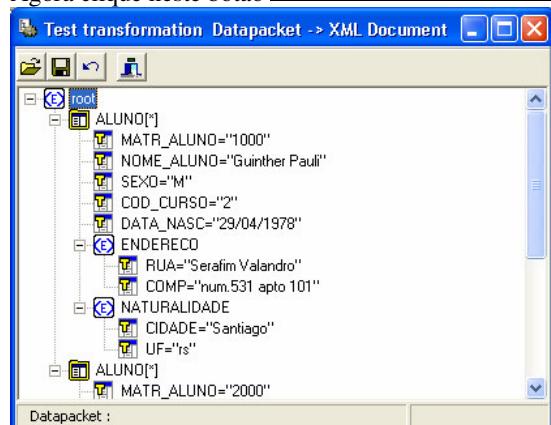


Figura. Configurando propriedades de um nó

Feche o *preview* e na janela principal do *XML Mapper* clique em *File|Save|Transformation*. Salve o arquivo gerado como *datapacket_to_xml*. Agora selecione a direção XML to *Datapacket*, porém com a opção *Insert Delta* para *Create Datapacket As*.

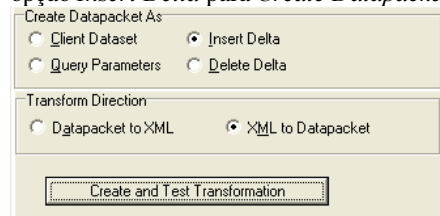


Figura. Insert Delta

Clique no botão para testar a transformação. Feche o *preview* e na janela principal do *XML Mapper* clique em *File|Save|Transformation*. Salve o arquivo gerado como *xml_to_interbase.xtr*

Usando documentos XML como Banco de Dados

Inicie um nova aplicação no Delphi 7. Clique em *File|Save All* e dê o nome de “uMainFrm.pas” para a unit e “XMLDataBase.dpr” para o projeto. Coloque no formulário um componente *XMLTransformProvider* da paleta *Data Access*, e também um *ClientDataSet*.

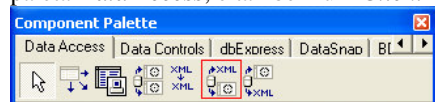


Figura. Componentes de transformação XML.

Selecione o componente *XMLTransformProvider* e configure as propriedades a seguir:

- *TransformRead.TransformationFile* = selecione o arquivo *xml_to_datapacket.xtr*;
- *TransformWrite.TransformationFile* = selecione o arquivo *datapacket_to_xml.xtr*;
- *XMLDataFile* = selecione o arquivo fonte dos dados, que é *alunos.xml*

Selecione o componente *ClientDataSet* e aponte sua propriedade *ProviderName* para *XMLTransformProvider1*. Ative o *ClientDataSet*, dê um duplo clique nele e pressione CTRL+F para adicionar os campos *TFields*.

Arraste os campos *TFields* para o formulário. Coloque no formulário um componente *DBNavigator* e configure seu *DataSource*. No evento *AfterPost* do *ClientDataSet* digite o seguinte:

```
ClientDataSet1.ApplyUpdates(0);
```

Execute a aplicação e insira um registro. Verifique se o mesmo foi inserido no documento XML original.

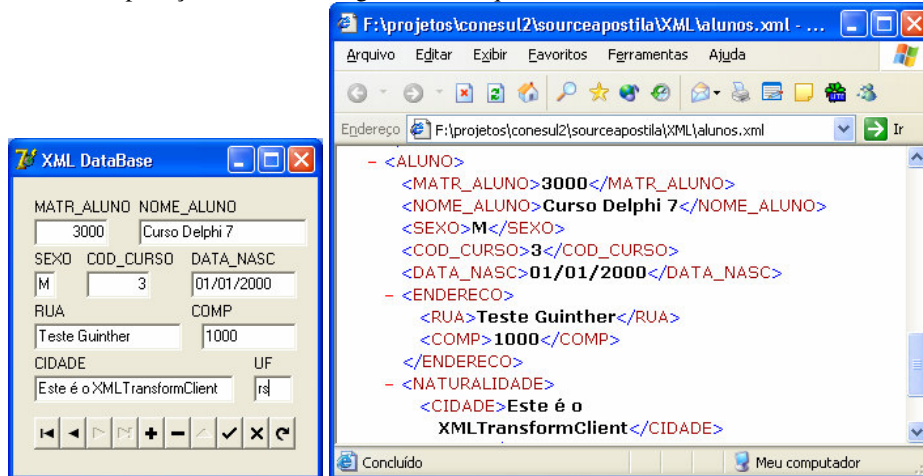


Figura. Usando XMLTranformProvider para usar um documento XML como Banco de Dados

A relação entre os componentes está representada no esquema abaixo:



Figura. Funcionamento do XMLTransformProvider

Aplicações B2B - Transferência de Dados e Serviços usando XML

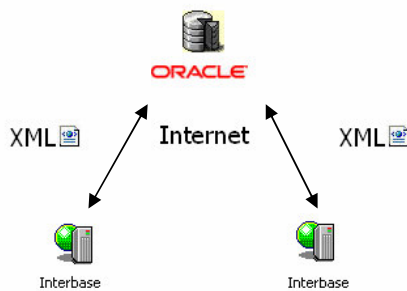


Figura. Aplicações B2B – troca de informações e serviços entre empresas através de XML

Primeiro vamos alterar a estrutura do nosso banco de dados utilizado no módulo de *dbExpress*. Abra o *IBConsole*, conecte no banco ESCOLA, abra o *Interactive SQL* e digite:

```
alter table ALUNOS
add RUA varchar(40), add COMP varchar(20), add CIDADE varchar(40), add UF char(2);
```

Crie uma nova aplicação no Delphi 7. Clique em *File|Save All*. Salve a unit como “uMainFrm.pas” e o projeto como “B2BApp.dpr”. Coloque no formulário um componente *XMLTransformClient* da paleta *Data Access*, e também um *DataSetProvider*.



Figura. Usando o XMLTransformClient

Da paleta *dbExpress* coloque um *SQLConnection*, configurando sua propriedade *ConnectionName* para *ESCOLA*, *LoginPrompt* para *False* e *Connected* para *True*. Coloque também um *SQLDataSet*, configurando seu *SQLConnection* para *SQLConnection1* e *CommandText* para “select * from ALUNOS”.

Aponte a propriedade *DataSet* de *DataSetProvider1* para *SQLDataSet1*. Aponte a propriedade *ProviderName* de *XMLTransformClient1* para *DataSetProvider1*. Abra a propriedade *TransformApplyUpdates.TransformationFile* de *XMLTransformClient1* e escolha o arquivo *xml_to_interbase.xtr* criado anteriormente no *XML Mapper*.

A relação entre os componentes está representada no esquema abaixo:



Figura. Componentes dbExpress e XMLTransformClient

Coloque no formulário um botão com o *Caption* de “Apply XML to DB” e um *OpenDialog* (paleta *Dialogs*). No evento *OnClick* do botão digite:

```

with TStringList.Create do
  if OpenDialog1.execute then
    begin
      LoadFromFile(OpenDialog1.FileName);
      XMLTransformClient1.ApplyUpdates(Text, '', -1);
      free;
    end;
  end;

```

Execute a aplicação, clique no botão, escolha o arquivo *alunos.xml*. Abra o *IBConsole* e verifique se eles foram inseridos.

MATR_ALUNO	NOME_ALUNO	DATA_NASC	SEXO	COD_CI	RUA	COMP	CIDADE	UF
8	LEONARDO	01/01/1675	M	10	<null>	<null>	<null>	<null>
9	GLADSTONE	01/01/1974	M	3	<null>	<null>	<null>	<null>
10	DANILON	01/01/1984	M	10	<null>	<null>	<null>	<null>
1000	Guinther Pauli	29/04/1978	M	2	Serafim Valandro	num.531 apto 101	Santiago	rs
2000	Rudolfo Pauli	14/05/1979	M	3	Serafim Valandro	num.531 apto 101	Santiago	rs
3000	Curso Delphi 7	01/01/2000	M	3	Teste Guinther	1000	Este é o XMLTrar	rs

Figura. Dados foram inseridos no banco de Dados Interbase a a partir de de um XMLTransformClient

Retirando dados do Interbase e distribuindo em formato XML (B2B)

Antes nós recebemos um documento XML, fizemos a transformação para *DataPacaket* e lançamos os dados no servidor SQL. Agora vamos fazer o processo inverso, vamos extrair os dados do Interbase e transformar os registros em informações formatadas em XML.

Para isso, ainda no mesmo exemplo anterior, basta apontar a propriedade *TransformGetData.TransformationFile* para o arquivo de transformação *datapacket_to_xml.xtr* criado anteriormente no *XML Mapper*.

Coloque no formulário outro botão com o *Caption* de “Get XML Data”. No seu evento *OnClick* digite:

```

with TStringList.create do
begin
  text:=XMLTransformClient1.GetDataAsXml('');
  SaveToFile('XML_from_ib.xml');
  free;
end;

```

Execute a aplicação e clique no botão. Abra o Explorer do Windows e verifique o arquivo *XML_from_ib.xml*. Lá deve estar toda a tabela do banco de dados extraída em formato XML.

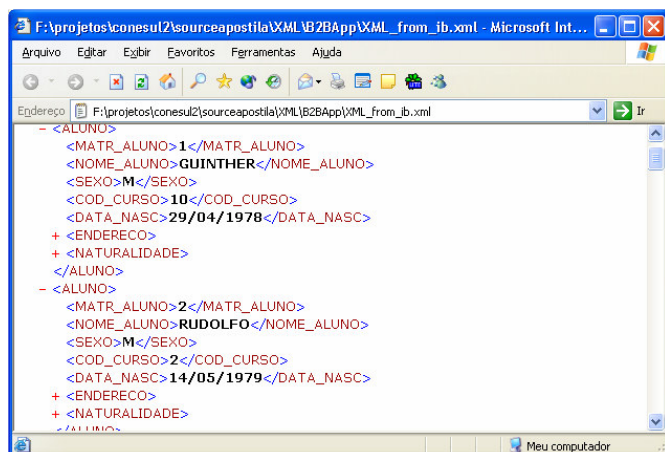


Figura. Dados extraídos no Interbase transformados em XML

Endereço para download dos exemplos

<http://codecentral.borland.com/codecentral/ccweb.exe/author?authorid=222668>