Configuração do acesso a dados (via ODBC).

A base de dados *Projecto1* foi criada como resultado da execução dos scripts exemplo. Em *Projecto1* podemos encontrar diversos elementos criados a partir dos scripts (tables, stored procedures, views, triggers, e dados em algumas tabelas). A base de dados *Projecto1* pode ser explorada usando, por exemplo, o *SQL Server Enterprise Manager*.

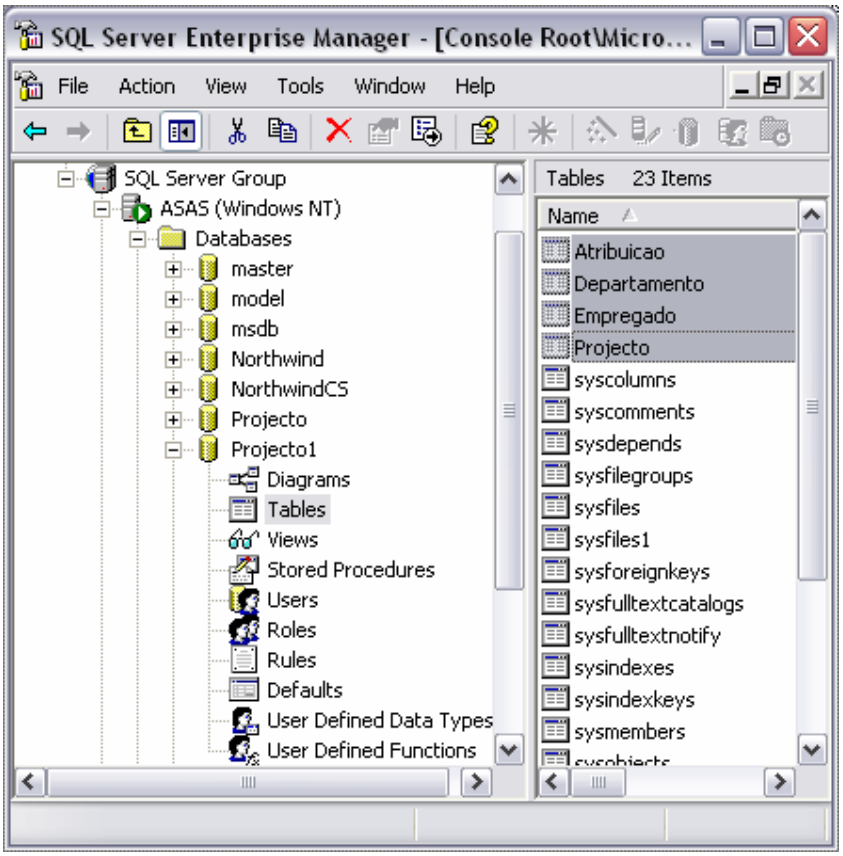

Figure 1 – Utilizar o SQL Server Enterprise Manager para explorar a base de dados *Projecto1*

A exploração da base de dados também pode ser feita via aplicações desenvolvidas usando linguagens de programação: Delphi, C/C++, Java, e outras. Para a disciplina de Bases de Dados a linguagem de programação seleccionada foi o Java, como o ambiente de desenvolvimento da Borland, mais concretamente, o JBuilder X Foundation.

Para podermos desenvolver aplicações que "falem" com a base de dados *Projecto1* é necessário configurar o acesso. No caso do servidor de bases de dados SQL Server 2000, o acesso pode fazer-se via ODBC.

As aplicações JBuilder comunicam com os servidores de bases de dados usando a API JDBC. No caso do acesso a dados ser via ODBC, é necessário fazer um "mapeamento" JDBC-ODBC.

Configurar a origem de dados via ODBC.

Executar a aplicação de configuração de acesso: Control Panel + Administrative Tools + Data Sources (ODBC).

Figure 2 – Configurar a origem de dados (ODBC)

As figuras Figure 3 a Figure 10 exemplificam o processo de configuração.

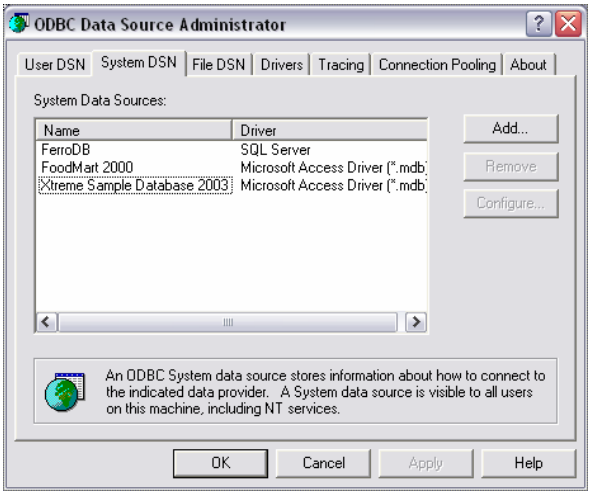

Figure 3 – Escolher System DSN (se o utilizador tiver privilégios para isso) e adicionar...

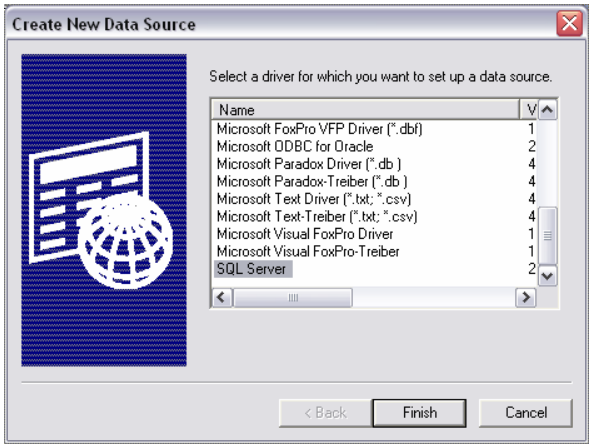

Figure 4 – Escolher o driver apropriado: SQL Server

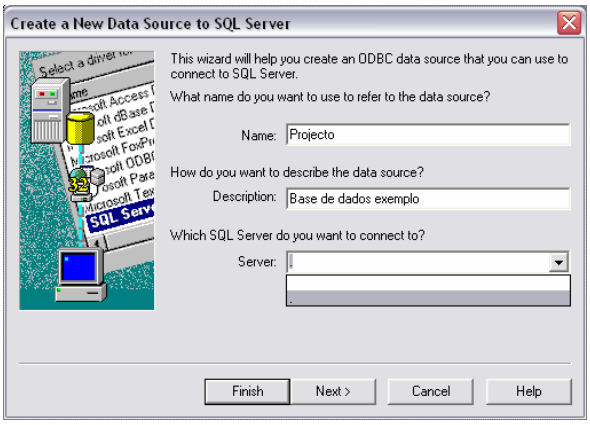

Figure 5 – Criar uma nova origem de dados. O nome dado à origem de dados, vai ser usado na aplicação desenvolvida no JBuilder!

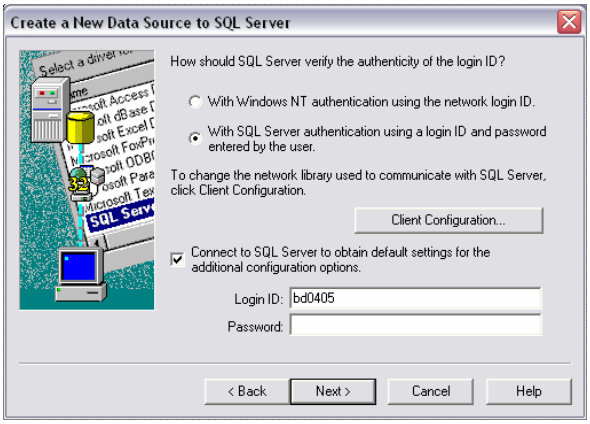

Figure 6 - Autenticação...

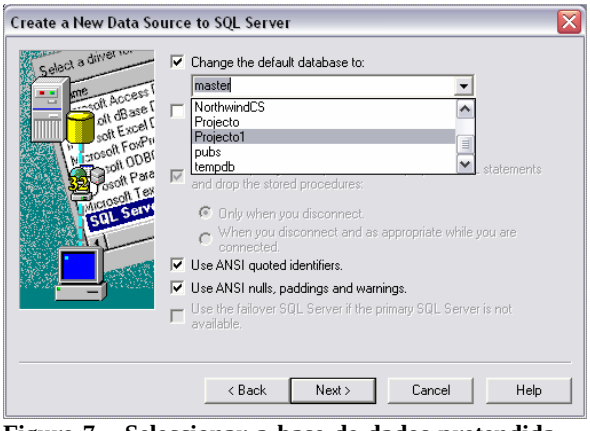

Figure 7 – Seleccionar a base de dados pretendida. Neste exemplo *Projecto1* é a base de dados criada pela execução dos scripts

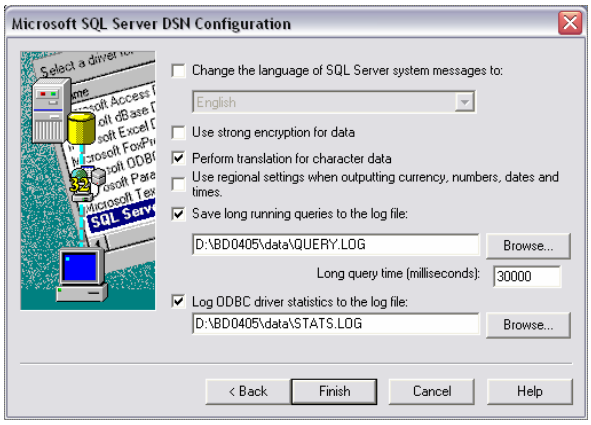

Figure 8 – Log associado ao acesso

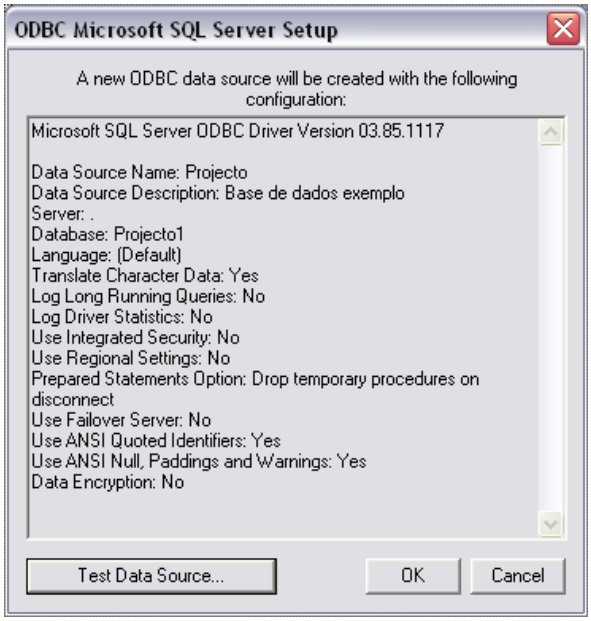

Figure 9 - Resumo das características

Figure 10 – Teste da origem de dados

Developing Database Applications JBuilder® 2005

Developing Database Applications provides information on using JBuilder's DataExpress database functionality to develop database applications. It also describes how to use dbSwing components to create a user interface (UI) for your application.

Be sure to check for documentation additions and updates at http://www.borland.com/techpubs/jbuilder.

Understanding JBuilder database applications

A database application is any application that accesses stored data and allows you to view and perhaps modify or manipulate that data. In most cases, the data is stored in a database. However, data can also be stored in files as text, or in some other format. JBuilder allows you to access this information and manipulate it using properties, methods, and events defined in the DataSet packages of the DataExpress Component Library in conjunction with the dbSwing package.

A database application that requests information from a data source such as a database is known as a client application. A DBMS (Database Management System) that handles data requests from various clients is known as a database server.

Database application architecture

JBuilder's DataExpress architecture is focused on building all-Java client-server applications, applets, servlets, and JavaServer Pages (JSP) for the inter- or intranet. Because applications you build in JBuilder are all-Java at run time, they are crossplatform.

JBuilder applications communicate with database servers through the JDBC API, the Sun database connectivity specification. JDBC is the all-Java industry standard API for accessing and manipulating database data. JBuilder database applications can connect to any database that has a JDBC driver.

DataExpress is a package, *com.borland.dx.dataset*, of Borland classes and interfaces that provide basic data access. This package also defines base provider and resolver classes as well as an abstract DataSet class that is extended to other DataSet objects. These classes provide access to information stored in databases and other data sources. This package includes functionality covering the three main phases of data handling:

- **Providing**
  General functionality to obtain data and manage local data sets. (JDBC specific connections to remote servers are handled by classes in the *com.borland.dx.sql*.dataset package.)
- **Manipulation**
  Navigation and editing of the data locally.
- **Resolving**
  General routines for the updating of data from the local DataSet back to the original source of the data. (Resolving data changes to remote servers through JDBC is handled by classes in the *com.borland.dx.sql.dataset* package.)

The core functionality required for data connectivity is contained in the *com.borland.dx.dataset*, *com.borland.dx.sql.dataset*, and *com.borland.datastor*e packages. The components in these packages encapsulate both the connection between the application and its source of the data, as well as the behavior needed to manipulate the data. The features provided by these packages include that of database connectivity as well as data set functionality.

The following diagram illustrates a typical database application and the layers from the client JBuilder DataExpress database application to the data source:

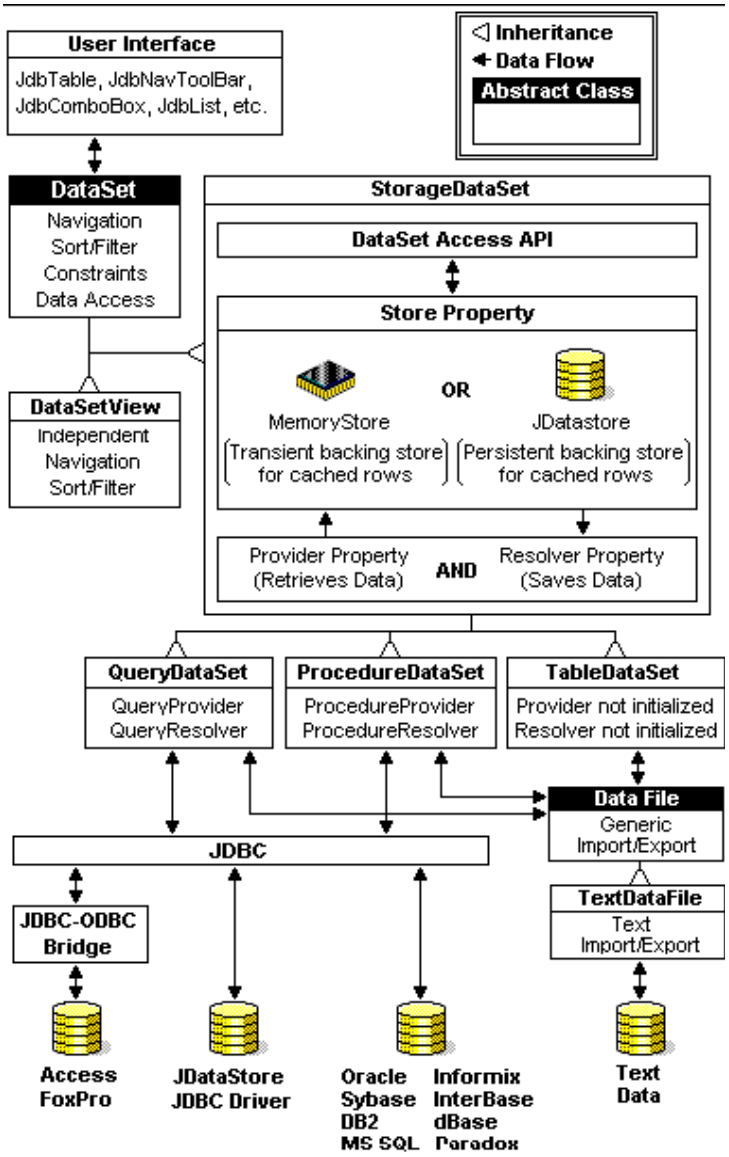

Figure 11 Diagram of a typical database application

dbSwing

The dbSwing package allows you to build database applications that take advantage of the Java Swing component architecture. In addition to pre-built, data-aware subclasses of most Swing components, dbSwing also includes several utility components designed specifically for use in developing DataExpress and JDataStore-based applications.

To create a database application, you first need to connect to a database and provide data to a DataSet.

To use the data-aware dbSwing components,

1. Open the Frame file, and select the Design tab.
2. Select one of the dbSwing pages: dbSwing, More dbSwing, or dbSwing Models.
3. Click a component on the component palette, and click in the UI designer to place the component in the application.
4. Select the component in the component tree or the UI designer. Depending on the type of component, and the layout property for the contentPane containing the component, the designer displays black sizing nibs on the edges of a selected component.

   Some of the component's (JdbNavToolBar and JdbStatusLabel) automatically bind to whichever data set has focus. For others (like JdbTable), set the component's dataSet and/or columnName properties in the Inspector to bind the component to an instantiated DataSet.

The following list contains a few of the dbSwing components available from the dbSwing page of the component palette:

- TableScrollPane
- JdbTable
- JdbNavToolBar
- JdbStatusLabel
- JdbTextArea
- JdbComboBox
- JdbLabel
- JdbList
- JdbTextPane
- JdbTextField

With increased functionality and data-aware capabilities, dbSwing offers significant advantages over Swing. Also, dbSwing is entirely lightweight, provides look-and-feel support for multiple platforms, and has strong conformance to Swing standards. Using dbSwing components, you can be sure all your components are lightweight.

Connecting to a database

Sun worked in conjunction with database and database tool vendors to create a DBMS independent API. Like ODBC (Microsoft's rough equivalent to JDBC), JDBC is based on the X/Open SQL Call Level Interface (CLI). Some of the differences between JDBC and ODBC are,

- JDBC is an all Java API that is truly cross platform. ODBC is a C language interface that must be implemented natively. Most implementations run only on Microsoft platforms.
- Most ODBC drivers require installation of a complex set of code modules and registry settings on client workstations. JDBC is an all Java implementation that can be executed directly from a local or centralized remote server. JDBC allows for much simpler maintenance and deployment than ODBC.

JDBC is endorsed by leading database, connectivity, and tools vendors including Oracle, Sybase, Informix, InterBase, DB2. Several vendors, including Borland, have JDBC drivers. Existing ODBC drivers can be utilized by way of the JDBC-ODBC bridge provided by Sun. Using the JDBC-ODBC bridge is not an ideal solution since it requires the installation of ODBC drivers and registry entries. ODBC drivers are also implemented natively which compromises cross-platform support and applet security.
JBuilder DataExpress components are implemented using the Sun database connectivity (JDBC) Application Programmer Interface (API). To create a Java data application, the Sun JDBC sql package must be accessible before you can start creating your data application. If your connection to your database server is through an ODBC driver, you also need the Sun JDBC-ODBC bridge software.
For more information about JDBC or the JDBC-ODBC bridge, visit the JDBC Database
Access API web site at http://java.sun.com/products/jdbc/.

Connecting to databases

You can connect JBuilder applications to remote or local SQL databases, or to databases created with other Borland applications such as C++ Builder or Delphi.
To connect to a remote SQL database, you need either of the following:

- A JDBC driver for your server. Some versions of JBuilder include JDBC drivers. One of these drivers is InterClient. Check the Borland web site at http://www.borland.com/jbuilder/ for availability of JDBC drivers in your edition of JBuilder or contact the technical support department of your server software company for availability of JDBC drivers.
- An ODBC-based driver for your server that you use with the JDBC-ODBC bridge software.

Note The ODBC driver is a non-portable DLL. This is sufficient for local development, but won't work for applets or other all-Java solutions.
When connecting to local, non-SQL databases such as Paradox or Visual dBASE, use an ODBC driver appropriate for the table type and level you are accessing in conjunction with the JDBC-ODBC bridge software.

Note When you no longer need a Database connection, you should explicitly call the Database.closeConnection() method in your application. This ensures that the JDBC connection is not held open when it is not needed and allows the JDBC connection instance to be garbage collected.

Adding a Database component to your application

The `Database` component is a JDBC-specific component that manages a JDBC connection. To access data using a `QueryDataSet` or a `ProcedureDataSet` component, you must set the database property of the component to an instantiated `Database` component. Multiple data sets can share the same database, and often will.

In a real world database application, you would probably place the `Database` component in a data module. Doing so allows all applications that access the database to have a common connection.

To add the `Database` component to your application,

1. Create a new project and application files using the Application wizard. (You can optionally follow these instructions to add data connectivity to an existing project and application.) To create a new project and application files:
   a) Choose File|Close from the JBuilder menu to close existing applications.
   If you do not do this step before you do the next step, the new application files will be added to the existing project.
   b) Choose File|New and double-click the Application icon to start the Application wizard.
   Accept or modify the default settings to suit your preferences.
2. Open the UI designer by selecting the Frame file (for example, `Frame1.java`) in the content pane, then select the Design tab at the bottom of the IDE.
3. Select the DataExpress page on the component palette, and click the Database component.
4. Click anywhere in the designer window to add the Database component to your application.

This adds the following line of code to the Frame class:

```
Database database1 = new Database();
```

Setting Database connection properties

The `Database connection` property specifies the JDBC driver, connection URL, user name, and password. The JDBC connection URL is the JDBC method for specifying the location of a JDBC data provider (for example, SQL server). It contains all the information necessary for making a successful connection, including user name and password.
You can access the `ConnectionDescriptor` object programmatically, or you can set connection properties through the Inspector. If you access the `ConnectionDescriptor` programmatically, follow these guidelines:

- If you set `promptPassword` to `true`, you should also call `openConnection`() for your database. `openConnection`() determines when the password dialog is displayed and when the database connection is made.
- Get user name and password information as soon as the application opens. To do this, call `openConnection`() at the end of the main frame's `jbInit`() method.

  If you don't explicitly open the connection, it will try to open when a component or data set first needs data.

Aplicação exemplo

Iniciar o *JBuilder X Foundation* e usar *File/New…* e seleccionar *Application.*

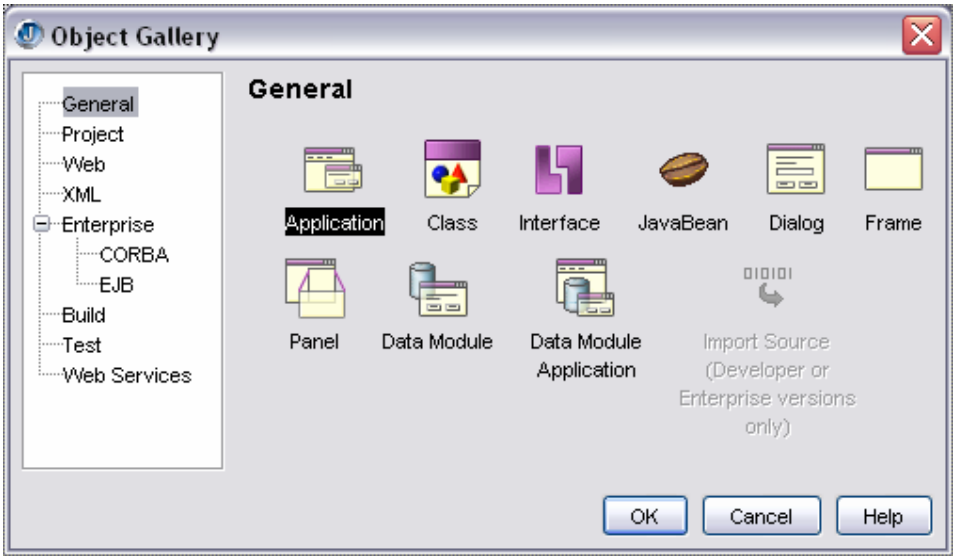

Figure 12 – Criar uma nova aplicação

Para obter uma aplicação simples basta seguir os Wizard, ver figuras Figure 13 a Figure 16.

Figure 13 – Indicar o nome e a directoria

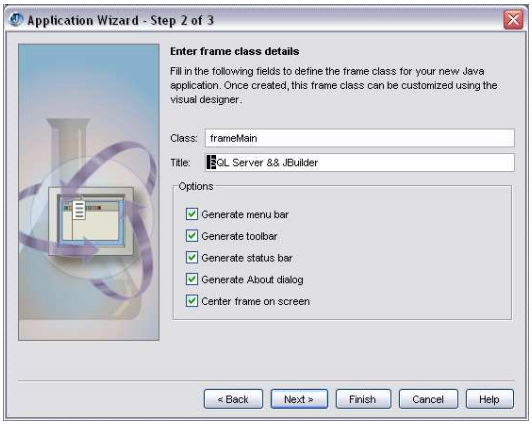

Figure 15 – Indicar o nome da frame e os menus a incluir

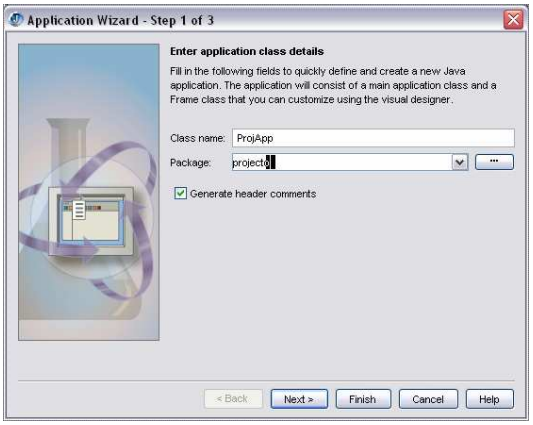

Figure 14 - Indicar a Class name & Package

Figure 16 – Indicar um nome para a configuração

Aceder à base de dados

Após termos executado (F9) a aplicação criada seguindo o Wizard, podemos incluir algumas componentes de modo a aproximar a nossa aplicação ao pretendido: aceder aos dados da base de dados *Projecto1*.

1º Passo: Incluir duas componentes na frame *frameMain*.

- Colocar a frame em primeiro plano (em modo de design);
- Adicionar uma componente *TableScrollPane*:
  - Seleccionar a palete *dbSwing* na palete de componentes;
  - Seleccionar (click) a componente *TableScrollPane* e fazer click na fram;
- Adicionar uma componente:
  - Seleccionar a palete *dbSwing* na palete de componentes;
  - Seleccionar (click) a componente *JdbTable* e fazer click na frame;
  - Dar um nome apropriado à componente, por exemplo dbtDepartamentos.

No final do 1º passo devemos ter uma grelha sobre a nossa frame. Esta grelha (*JdbTable*) vai-nos permitir visualizar dados, por exemplo os dados da tabela Departamento. Estes dados hão-de chegar à grelha usando outras componentes, nomeadamente, uma *Database* (componente para aceder ao servidor da base de dados) e uma *QueryDataSet* (componente onde podemos especificar uma query para aceder aos dados).

2º Passo: Incluir uma componente *Database*.

- Colocar a frame *frameMain* em primeiro plano (em modo de design);
- Adicionar uma componente *Database*:
  - Seleccionar a palete *DataExpress* na palete de componentes;
  - Seleccionar (click) a componente *Database* e fazer click na frame *frameMain*;
  - Dar um nome apropriado para esta componente, por exemplo, dbProjecto.
- Especificar as propriedades da componente dbProjecto:
  - Especificar as propriedades da ligação (Figure 17e Figure 18). Note-se em particular o URL usado: jdbc:odbc:**Projecto**.

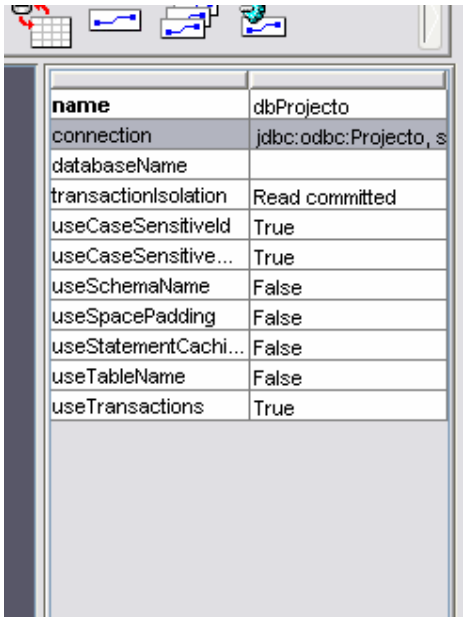

Figure 17 – Especificar as propriedades da ligação

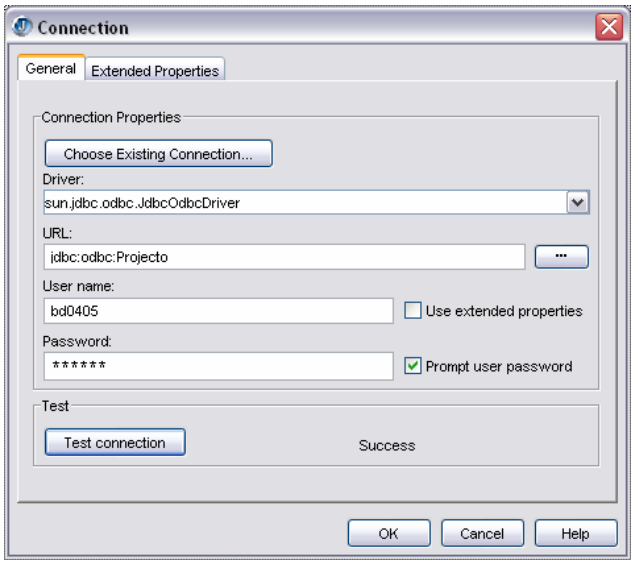

Figure 18 – Escolher o driver e a base de dados que se pretende usar. Indicar o user name e a password e testar a ligação. Seleccionar a opção "Prompt user password"

A Figure 19 ilustra a aplicação exemplo

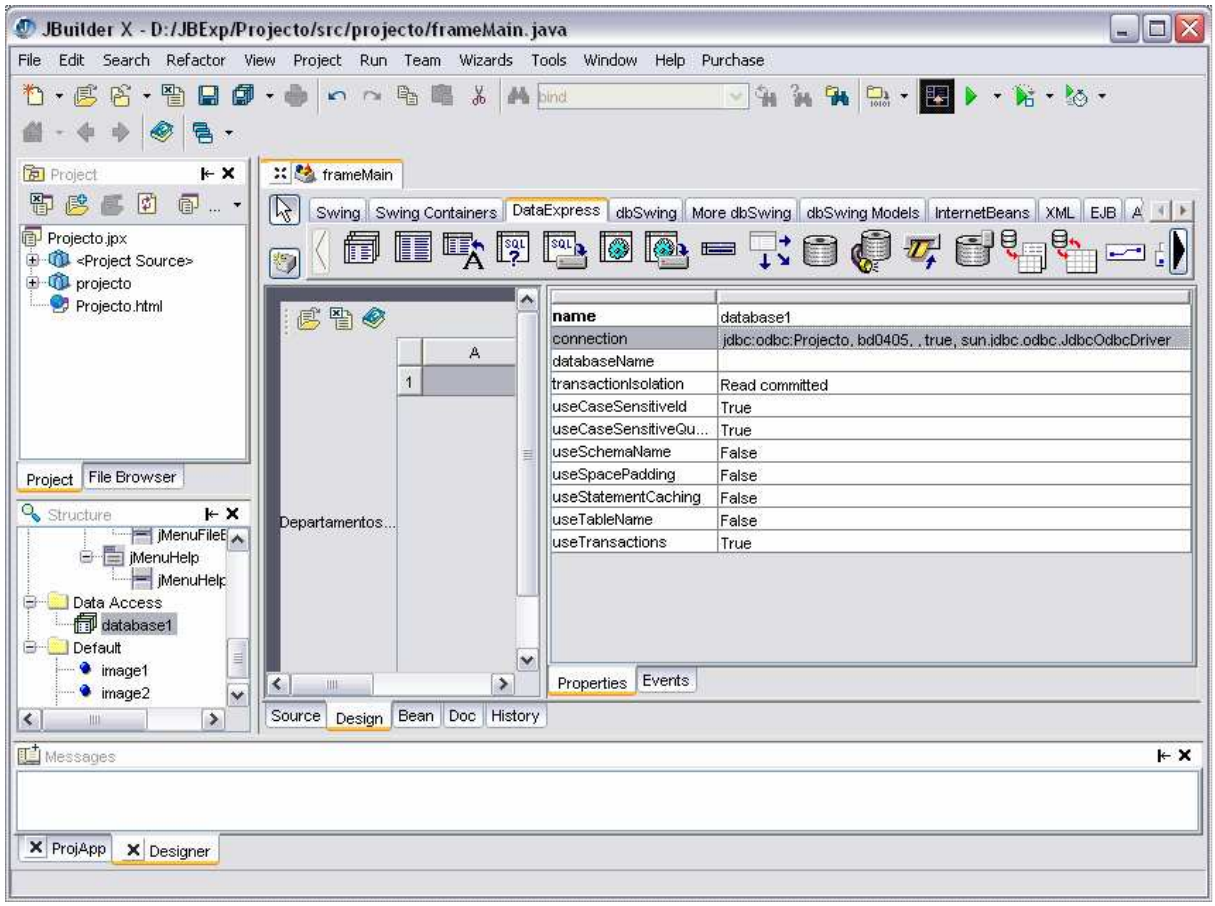

Figure 19 – Aplicação com uma componente Database (database1)

Uma vez configurado o acesso a dados, e encapsulado, numa componente Database (dbProjecto), podemos aceder aos dados. Por exemplo, para mostrar todas os departamentos basta executar a instrução "Select * from departamento". A componente *QueryDataSet* pode ser usada para este fim.

3º Passo: Incluir uma componente *QueryDataSet*.

- Colocar a frame *frameMain* em primeiro plano (em modo de design);
- Adicionar uma componente *QueryDataSet*:
  - Seleccionar a palete *DataExpress* na palete de componentes;
  - Seleccionar (click) a componente *QueryDataSet* e fazer click na frame *frameMain*;
  - Dar um nome apropriado para esta componente, por exemplo, qdsDepSelect.
- Especificar as propriedades da componente qdsDepSelect:
  - Especificar a instrução de interrogação e fazer a ligação à componente dbProjecto (Figure 20 e Figure 21).

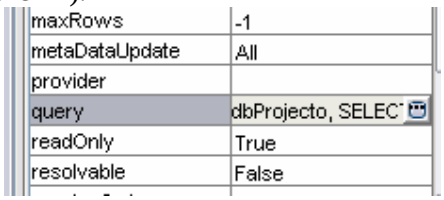

Figure 20 - Seleccionar query...

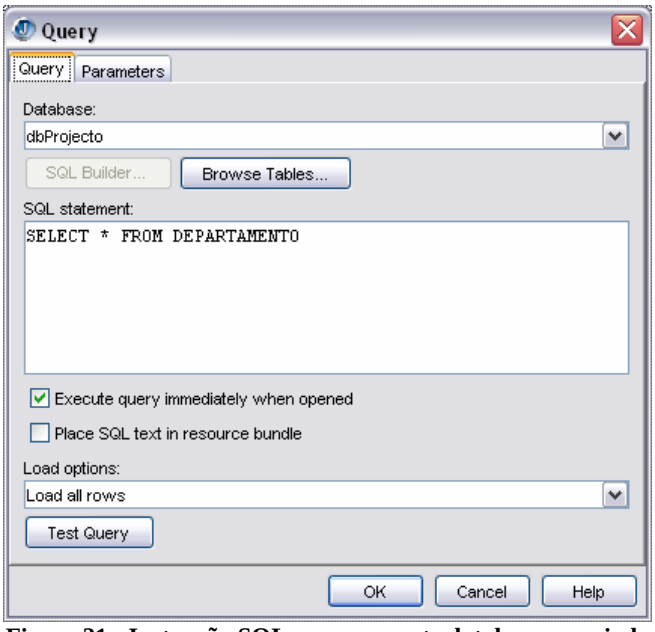

Figure 21 - Instrução SQL e componente database associada

4º Passo: Encaminhar os resultados para a componente JdbTable:

- Colocar a frame *frameMain* em primeiro plano (em modo de design);
- Seleccionar a componente dbtDepartamentos:
  - Seleccionar a propriedade *dataSet* e escolher a componente qdsDepSelect.

e pronto…temos dados (ver Figure 22):

	DepNum	Nome	Local
1	1	Camarote	Camarate
2	2	Informática	Covilhã
3	3	Produção	Guarda
4	4	Comercial	Lisboa

Figure 22 - Dados obtidos pela query qdsDepSelect e encaminhados para a grelha dbtDepartamentos

Nesta fase podemos executar a aplicação e verificar que esta se comporta de forma apropriada. De seguida vamos incluir mais alguma funcionalidade na aplicação, nomeadamente, vamos inserir/alterar/eliminar departamentos e dar a possibilidade ao utilizador para obter dados mais recentes. Note-se que vários utilizadores podem estar a actualizar a base de dados, donde esta vai sendo alterada. Logo a nossa grelha pode estar a mostrar dados fantasma…

Incluir o menu Departamento

A primeira modificação à aplicação incluir um menu Departamento e as respectivas opções (Inserir, Alterar, Eliminar e Refrescar).

A Figure 23 ilustra o efeito pretendido.

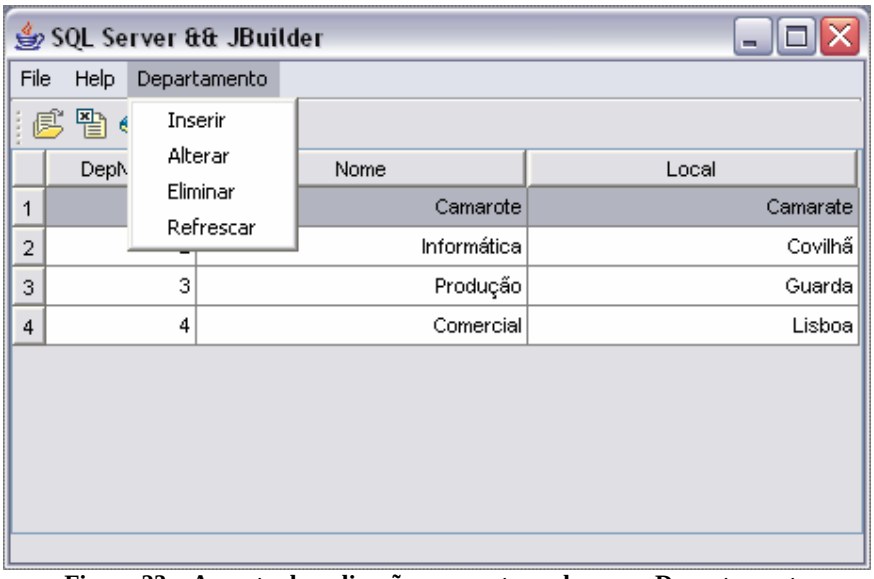

Figure 23 – Aspecto da aplicação a executar e do menu Departamento

Quando o utilizador seleccionar a opção Inserir deve-lhe ser dada a oportunidade de inserir um novo departamento. A Figure 24 ilustra este aspecto da aplicação.

Figure 24 – Inserir departamento

O código do departamento é colocado automaticamente (!) com o número do departamento a inserir. Como vários utilizadores podem executar em simultâneo esta aplicação (ou outra com efeito semelhante), o código sugerido (5) pode já ter sido inserido, e nesse caso teríamos um problema. Na aplicação exemplo, vamos mostrar uma mensagem de erro e dar uma oportunidade ao utilizador para tentar novamente (Figure 25.

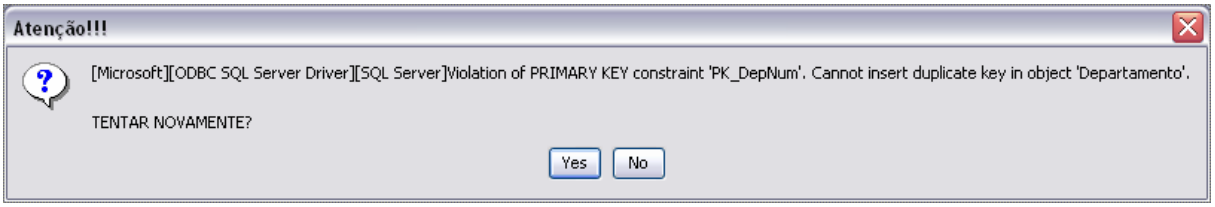

Figure 25 - Erro na inserção de um departamento: chave primária violada

Portanto, a aplicação deve estar preparada para tratar deste problema, procurando, por exemplo, um novo código.

Inserir um departamento

1. Usar *File/New…* e seleccionar *Dialog*.
2. Incluir etiquetas (JLabel), caixas de texto (TTextField) e botões (JButton) de modo a dar um aspecto adequado ao formulário (JDialog).
3. Dar nomes apropriados às componentes e guardar o trabalho, usar por exemplo o nome dlgDepInserir.

4. Criar um método, por exemplo, *public int showDepInsAlt(int DepNum, StringBuffer Nome, StringBuffer Local)* que permita mostrar o diálogo e mostrar os valores do DepNum, Nome e Local do departamento. Este método deve devolver 0 se o utilizador tiver escolhido o botão OK para fechar o diálogo. O extracto de código seguinte ilustra esta função:

```
//.....................................................................
// showDepInsAlt(int DepNum, String Nome, String Local)
//   Parameters:
//      [in]:
//            DepNum - the number of department.
//      [inout]:
//            Nome   - string, the name of department
//            Local  - string, Location
//   Result: Integer
//      0  - Button Ok has clicked and (Nome or Local modified)
//      >0 - Button Cancel has clicked
//      <0 - An error has ocurred
//
// purpose: get the department Name and Location.
//.....................................................................

 public int showDepInsAlt(int DepNum, StringBuffer Nome, StringBuffer Local){

   jtfDepCod.setText( Integer.toString( DepNum));
   jtfNome.setText(Nome.toString());
   jtfLocal.setText(Local.toString());
   this.show();

   jtfNome.setText(jtfNome.getText().trim());
   jtfLocal.setText(jtfLocal.getText().trim());

   if (isClickedjbtOK && (Nome.toString() != jtfNome.getText()) &&
       (Local.toString() != jtfLocal.getText())){
     Nome.append(jtfNome.getText());
     Local.append(jtfLocal.getText());
     return 0;
   }
   else
     return 1;
  }
```

5. Quando o utilizador escolher a opção Departamento/Inserir, deve-lhe ser mostrado o diálogo, por exemplo, usando o código:

```
void miDepInserir_actionPerformed(ActionEvent e) {
   dlgDepInserir dlg = new dlgDepInserir(this, "Inserir departamento", true);

   StringBuffer Nome = new StringBuffer("");
   StringBuffer Local = new StringBuffer("");
   int nextDep;

   while (true) {
      nextDep = GetNextDepNum(); // get the next depnum

      if (dlg.showDepInsAlt(nextDep, Nome, Local) == 0) {
        try {
          DepInserir(nextDep, Nome.toString(), Local.toString());
```

6. Obter o número de departamento: função GetNextDepNum.

```
//.....................................................................
// GetNextDepNum
// input:  none
// output: int
// purpose: get the next department number.
//.....................................................................
  private int GetNextDepNum() {
    // try to execute the query
    try {
      qdsDepNext.executeQuery();
    }
    catch (Exception ex) {
      return -1;
```

```
    }

    // gather the results
    DataRow resultRow = new DataRow(qdsDepNext);
    qdsDepNext.getDataRow(resultRow);

    // return the next dep number
    if (resultRow.isNull("NextDep"))
      return 1;
    else
      return 1 + resultRow.getInt("NextDep");
  }
```

Esta função emprega uma componente QueryDataSet, qdsDepNext, para obter o máximo DepNum da tabela Departamento. A instrução SQL usada é a seguinte: `"Select Max(DepNum) NextDep from Departamento";`

7. Inserir os dados introduzidos pelo utilizador na tabela Departamento:

```
//.....................................................................
// DepInserir
// input:
//     [in] DepNum - number of departamento
//     [in] Nome - name of departamento
//     [in] Local - location of departamento
// output: none
// purpose: insert a new raw in departamento table.
//.....................................................................
  private void DepInserir(int DepNum, String Nome, String Local) {
    // set the parameter raw values
    prDepValues.setInt("DepNum", DepNum);
    prDepValues.setString("Nome", Nome);
    prDepValues.setString("Local", Local);

    // Use a query provider ro execute a SQL Statement which don't
    // return a result set!
    QueryProvider qpDepInserir = new QueryProvider();
    qpDepInserir.executeStatement(dbProjecto,SQL_INS_DEP, prDepValues);

    qpDepInserir.closeStatement();
  }
```

Como esta operação não envolve trazer dados da base de dados, a componente a usar deve ser uma *QueryProvider* e não uma *QueryDataSet*. A instrução SQL usada é a seguinte: `Insert Into Departamento Values (:DepNum,:Nome,:Local)`. Os parâmetros (:DepNum, :Nome e :Local) são estabelecidos pelo emprego da componente ParameterRow (prDepValues).

Alterar um departamento

1. Associar código à selecção de Departamento/Alterar:

```
void miDepAlterar_actionPerformed(ActionEvent e) {
    dlgDepInserir dlg = new dlgDepInserir(this, "Alterar departamento", true);

…

    DataRow resultRow = new DataRow(qdsDepSelect);
    qdsDepSelect.getDataRow(resultRow);

    if (!resultRow.isAssignedNull("DepNum")) {
      int nextDep = resultRow.getInt("DepNum");
      StringBuffer Nome = new StringBuffer(resultRow.getString("Nome"));
      StringBuffer Local = new StringBuffer(resultRow.getString("Local"));

      if (dlg.showDepInsAlt(nextDep, Nome, Local) == 0) {
        try {
          DepAlterar(nextDep, Nome.toString(), Local.toString());
```

2. Vamos usar o mesmo formulário da operação Departamento/Inserir. Os dados a serem alterados correspondem ao departamento que estiver seleccionado na grelha dbtDepartamentos.

   Podemos usar um objecto DataRow para obtermos os dados pretendidos.

```
    DataRow resultRow = new DataRow(qdsDepSelect);
    qdsDepSelect.getDataRow(resultRow);
```

3. Alterar os dados do departamento:

```
//.....................................................................
// DepAlterar
// input:
//     [in] DepNum - number of departamento
//     [in] Nome - name of departamento
//     [in] Local - location of departamento
// output: none
// purpose: update a raw in departamento table, identified by DepNum.
//.....................................................................
  private void DepAlterar(int DepNum, String Nome, String Local) {
    // set the parameter raw values
/*
    prDepValues.setInt("DepNum", DepNum);
    prDepValues.setString("Nome", Nome);
    prDepValues.setString("Local", Local);

    QueryProvider qpDepAlterar = new QueryProvider();
    qpDepAlterar.executeStatement(dbProjecto,
          "Update Departamento\nSet Nome =:Nome,\n    Local =:Local\nWhere DepNum "
                                    + "= :DepNum", prDepValues);
    qpDepAlterar.closeStatement();
*/


   Connection _conn = dbProjecto.getJdbcConnection();
   try {

       // inserir despesa
       PreparedStatement sd  = _conn.prepareStatement(SQL_UPD_DEP);
       sd.setString(1, Nome);
       sd.setString(2, Local);
       sd.setInt(3, DepNum);
       sd.executeUpdate();
       sd.close();

     } catch (SQLException e) {
         System.err.println("ERRO: Inserir despesa - " + e.getMessage());
      }
  }
```

Podemos usar uma componente QueryProvider para executar a operação pretendida, análogo ao modo como foi feita a inserção (em comentário, sombreado mais escuro); ou sem recorrermos às componentes podemos preparar um *statement* Java, para efectuarmos a alteração dos dados.
A instrução SQL a ser enviada ao servidor é a seguinte: "`Update Departamento Set Nome =?, Local =? Where DepNum = ?`".