

Entendiendo ADO .NET con Mono y PostgreSQL (III)

por Martín Márquez xomalli@gmail.com

Introducción

ADO .NET es la evolución de la tecnología ADO (Active X Data Objects) que permite la comunicación de aplicaciones con bases de datos relacionales en sistemas operativos Microsoft, ADO fue diseñado para ser dependiente de la plataforma y trabajar de manera siempre conectada con la fuente de datos al momento de hacer operaciones, ADO .NET además de tener clases para trabajar con este modelo, presenta clases para trabajar con datos de una manera desconectada, esto es extrayendo los datos de la fuente mediante una consulta, desconectándose y manteniendo una copia en memoria, hacer la cantidad de operaciones necesarias y devolverlos a la fuente de datos de donde fueron tomados para mantener los cambios.

En este documento mostraremos mediante un formulario el uso de las clases de ADO .NET en un escenario de datos desconectado y como este modelo funciona con software libre y sobre sistemas operativos no Microsoft usando la plataforma .Net del proyecto mono <http://www.go-mono.com> y el servidor de bases de datos PostgreSQL <http://www.postgresql.org>.

La clase Dataset y el modelo desconectado

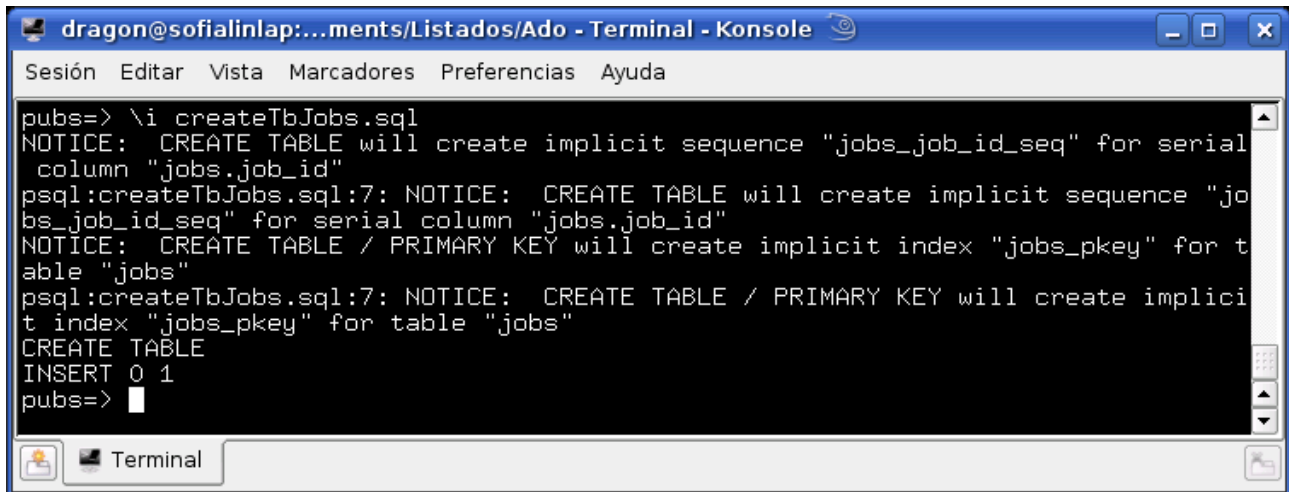
El modelo desconectado de ADO .NET se centra en la clase DataSet la cual es una representación en memoria de la estructura de datos similar a la estructura de una base de datos relacional.

Para nuestro ejemplo crearemos una script para la base de datos pubs (creada anteriormente).

Listado 3.1 Script para crear la tabla jobs con un registro

```
CREATE TABLE jobs (  
    job_id serial NOT NULL PRIMARY KEY,  
    job_desc character varying NOT NULL,  
    min_salary integer NOT NULL,  
    max_salary integer NOT NULL  
);  
INSERT INTO jobs(job_desc,min_salary,max_salary)  
VALUES ('Programador',1000,2000);
```

Ejecutamos el *script* en PostgreSQL para crear la tabla.



```
dragon@sofialinlap:...ments/Listados/Ado - Terminal - Konsole
Sesión Editar Vista Marcadores Preferencias Ayuda
pubs=> \i createTbJobs.sql
NOTICE: CREATE TABLE will create implicit sequence "jobs_job_id_seq" for serial
column "jobs.job_id"
psql:createTbJobs.sql:7: NOTICE: CREATE TABLE will create implicit sequence "jo
bs_job_id_seq" for serial column "jobs.job_id"
NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "jobs_pkey" for t
able "jobs"
psql:createTbJobs.sql:7: NOTICE: CREATE TABLE / PRIMARY KEY will create implici
t index "jobs_pkey" for table "jobs"
CREATE TABLE
INSERT 0 1
pubs=> █
```

Una vez con la tabla creada usaremos el siguiente listado para mostrar, agregar y eliminar datos usando las clases: NpgsqlDataAdapter, Dataset y CommandBuilder las cuales son esenciales para realizar operaciones en el modelo de datos desconectado y las explicaremos en el resto de este documento.

Listado 3.2 Formulario GTK# para demostrar el trabajo con datos de manera desconectada

```
using System;
using Gtk;
using System.Data;
using Npgsql;

namespace Godel.Listados {
class Listado3_4 : Gtk.Window {
private Label lbMsg = new Label("");
private Entry txtId = new Entry();
private Entry txtDescr = new Entry();
private Entry txtsalaryMin = new Entry();
private Entry txtsalaryMax = new Entry();
private Button btnNew = new Button(Stock.New);
private Button btnDel = new Button(Stock.Delete);
private Button btnCommit = new Button("Sincronizar");
private Button btnAdd = new Button(Stock.Add);
private Button btnCancel = new Button(Stock.Cancel);
private DataSet ds = null;
private NpgsqlDataAdapter dataAdapter = null;
private NpgsqlConnection conn = null;
private int numRecord = 0,regId = 0,maximumRecord = 0;
private HScrollbar navigatorBar = null;

public Listado3_4(): base("Listado 3.4"){
BorderWidth = 8;
SetDefaultSize(208,220);
this.DeleteEvent += new DeleteEventHandler(OnWindowDelete);
Frame frame = new Frame ("Registro de profesiones");
Add (frame);
VBox MainPanel = new VBox (false, 8);
frame.Add (MainPanel);
Table table = new Table (4, 2, false);
table.RowSpacing = 2;
table.ColumnSpacing = 2;
MainPanel.PackStart (table);
```

```

Label lbjob_id = new Label("id");
table.Attach(lbjob_id, 0, 1, 0, 1);
txtId.IsEditable = false;
table.Attach(txtId, 1, 2, 0, 1);
Label lbjob_desc = new Label("Descripcion");
table.Attach(lbjob_desc, 0, 1, 1, 2);
table.Attach(txtDescr, 1, 2, 1, 2);
Label lbmin_salary = new Label("Sueldo minino");
table.Attach(lbmin_salary, 0, 1, 2, 3);
table.Attach(txtsalaryMin, 1, 2, 2, 3);
Label lbmax_salary = new Label("Sueldo maximo");
table.Attach(lbmax_salary, 0, 1, 3, 4);
table.Attach(txtsalaryMax, 1, 2, 3, 4);
HGroupBox controlPanel = new HGroupBox();
controlPanel.Layout = ButtonBoxStyle.Start;
controlPanel.PackStart(btnNew, false, true, 8);
controlPanel.PackStart(btnDel, false, true, 8);
controlPanel.PackStart(btnCommit, false, true, 8);
MainPanel.PackStart(controlPanel, false, false, 0);
HGroupBox AddPanel = new HGroupBox();
AddPanel.Layout = ButtonBoxStyle.Start;
AddPanel.PackStart(btnAdd, false, true, 8);
AddPanel.PackStart(btnCancel, false, true, 8);
MainPanel.PackStart(AddPanel, false, false, 0);
Adjustment adj = new Adjustment(0, 0, 1, 1, 0, 0);
navigatorBar = new HScrollbar(adj);
HBox browsePanel = new HBox();
browsePanel.PackStart(navigatorBar, true, true, 8);
MainPanel.PackStart(browsePanel, false, false, 0);
MainPanel.PackStart(lbMsg, true, false, 0);
btnNew.Clicked += new EventHandler(btnClickedNew);
btnDel.Clicked += new EventHandler(btnClickedDel);
btnCommit.Clicked += new EventHandler(btnClickedCommit);
btnAdd.Clicked += new EventHandler(btnClickedAdd);
btnCancel.Clicked += new EventHandler(btnClickedCancel);
navigatorBar.ChangeValue += new
ChangeValueHandler(navigatorBar_ChangeValue);
ShowAll();
setAdd(false);
bindData();
}

public void OnWindowDelete(object o, DeleteEventArgs args) {
Application.Quit();
}

void btnClickedNew(object o, EventArgs args){
clearTexts();
regId = MAXIMUMUNRECORD;
regId++;
txtId.Text = regId.ToString();
setAdd(true);
}

void btnClickedDel(object o, EventArgs args){
if(navigatorBar.Value > 0){
DataRow r =
ds.Tables[0].Rows[Convert.ToInt32(navigatorBar.Value)];
r.Delete();
displayReg(ds, Convert.ToInt32(navigatorBar.Value - 1));
}
}

void btnClickedCommit(object o, EventArgs args){

```

```

        try {
            //abrimos la conexion al server
            conn.Open();
            //creamos el constructor de comandos
            NpgsqlCommandBuilder cmdBuilder = new
NpgsqlCommandBuilder(dataAdapter);
            int rowsAffected = dataAdapter.Update(ds,"jobs");
            using (Dialog dialog = new
MessageDialog(this,DialogFlags.Modal,MessageType.Info,ButtonsType.Ok,
                "(" + rowsAffected.ToString()+ ")
registro(s) afectado(s)",
                    0)) {
                dialog.Run ();
                dialog.Hide ();
            }
        }
        catch(NpgsqlException e){ lbMsg.Text = e.Message; }
        finally { conn.Close(); }
    }

    void btnClickedAdd(object o, EventArgs args){
        if(txtId.Text.Length > 0)
        {
            DataRow r = ds.Tables[0].NewRow();
            r["job_id"] = txtId.Text;
            r["job_desc"] = txtDescr.Text;
            r["min_salary"] = txtsalaryMax.Text;
            r["max_salary"] = txtsalaryMin.Text;
            ds.Tables[0].Rows.Add(r);
            displayReg(ds,MAXIMUNRECORD - 1);
            setAdd(false);
        }
        else { lbMsg.Text = "Teclee un ID"; }
    }

    void btnClickedCancel(object o, EventArgs args){
        setAdd(false);
        displayReg(ds,MAXIMUNRECORD-1);
    }

    private void setAdd(bool b){
        if(b){
            btnAdd.Show();
            btnCancel.Show();
            btnNew.Hide();
            btnDel.Hide();
            btnCommit.Hide();
        }
        else{
            btnAdd.Hide();
            btnCancel.Hide();
            btnNew.Show();
            btnDel.Show();
            btnCommit.Show();
        }

        txtDescr.IsEditable = b;
        txtsalaryMin.IsEditable = b;
        txtsalaryMax.IsEditable = b;
    }

    private void bindData(){
        conn = new NpgsqlConnection("Server=127.0.0.1;Port=5432;User
Id=postgres;Password=chikome;Database=pubs;");
        dataAdapter = new NpgsqlDataAdapter("SELECT

```

```

job_id,job_desc,min_salary,max_salary FROM jobs",conn);
    ds = new DataSet();
    try{ dataAdapter.Fill(ds,"jobs");}
    catch(NpgsqlException e){ lbMsg.Text = e.Message; }
    finally { conn.Close(); }
    displayReg(ds,numRecord);
}

private void displayReg(DataSet ds,int row){
    MAXIMUNRECORD = ds.Tables[0].Rows.Count;
    navigatorBar.SetRange(0,MAXIMUNRECORD - 1);
    DataTable tbJobs = ds.Tables[0];
    if(tbJobs.Rows.Count != 0){
        txtId.Text = Convert.ToString(tbJobs.Rows[row][0]);
        txtDescr.Text = Convert.ToString(tbJobs.Rows[row][1]);
        txtsalaryMin.Text = Convert.ToString(tbJobs.Rows[row][2]);
        txtsalaryMax.Text = Convert.ToString(tbJobs.Rows[row][3]);
    }
    lbMsg.Text = "Registro " + Convert.ToString(row + 1) + " de " +
MAXIMUNRECORD;
    navigatorBar.Value = row;
}

public int MAXIMUNRECORD{
    set{ maximumRecord = value; }
    get{ return maximumRecord; }
}

private void clearTexts(){
    txtId.Text = "";
    txtDescr.Text = "";
    txtsalaryMin.Text = "";
    txtsalaryMax.Text = "";
}

[STAThread]
public static void Main(string[] args){
    Application.Init();
    new Listado3_4();
    Application.Run();
}

private void navigatorBar_ChangeValue(object o, ChangeValueArgs args){
    numRecord = Convert.ToInt32(navigatorBar.Value);
    displayReg(ds,numRecord);
}
}
}

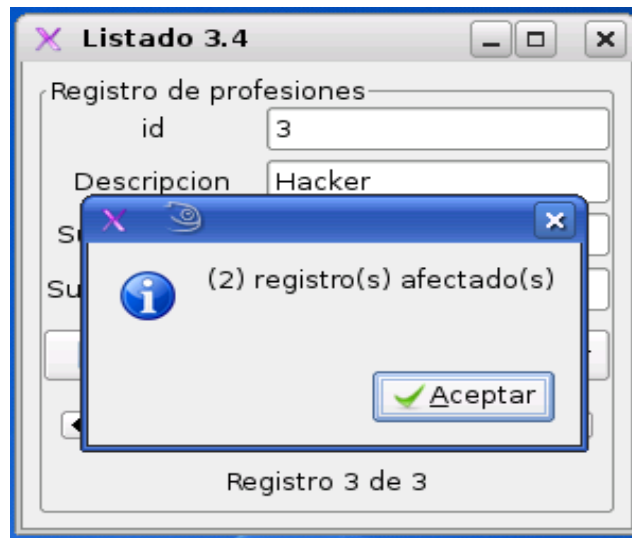
```

En la función *bindData()* usamos el método *fill()* de la clase *NpgsqlDataAdapter* para ejecutar la consulta SQL y almacenar los datos devueltos en la clase *DataSet*, el método *fill()* recibe como argumentos el nombre de la instancia *DataSet* y el nombre de la tabla de la cual proviene la información.

Una vez con la información en el *DataSet* podemos trabajar con los datos de manera desconectada, las funciones delegadas *btnClickedAdd* y *btnClickedDel* operan **exclusivamente** con los registros del almacenados en el objeto *DataRow* utilizando los métodos *Add()* y *Delete()* respectivamente.

El objeto *NpgsqlDataAdapter* además de hacer el llenado del *DataSet* es también responsable de que los cambios hechos sean aplicados al servidor de datos utilizando el método *Update()*, recuperando la cantidad de registros afectados en una variable de tipo entera.

```
int rowsAffected = dataAdapter.Update(ds, "jobs");
```



NpgsqlCommandBuilder construye los comandos INSERT, DELETE y UPDATE basandose en el esquema del DataSet, y que es importante para que se ejecuten estos comandos al momento de sincronizar.

```
NpgsqlCommandBuilder cmdBuilder = new NpgsqlCommandBuilder(dataAdapter);
```

La clase *NpgsqlDataAdapter* necesita una conexión física al servidor de la base de datos, si la conexión no esta abierta, la clase *NpgsqlDataAdapter* se encarga de abrirla y cerrarla, esta practica no es muy recomendable, lo mejor es abrir y cerrar la conexión manualmente sin olvidar ejecutarla dentro del bloque [try/catch/finally](#) para una mayor seguridad.

```

dragon@sofialinlap:~/...ments/Listados/Ado - Terminal - Konsol
Sesión  Editar  Vista  Marcadores  Preferencias  Ayuda
pubs=> select * from jobs;
 job_id | job_desc | min_salary | max_salary
-----+-----+-----+-----
      1 | Programador |      1000 |      2000
      2 | dbadmin    |       500 |       200
      3 | Hacker    |     20000 |     10000
(3 rows)
pubs=>

```

Conclusión

ADO .Net presenta capacidades muy potentes para el trabajo con bases de datos relacionales que son aprovechables dentro del ambiente GNU/Linux, mi intención fue mostrar en resumen las características más elementales y significativas de ADO .Net recomendando las siguientes referencias para una mayor profundidad en el tema.

Sobre material de **PostgreSQL** recomiendo el excelente sitio <http://www.postgresql.cl/> y para ADO .NET <http://npgsql.projects.postgresql.org/docs/manual/UserManual.html>

Los ejemplos pueden ser descargados de <http://www.humansharp.com/index.php?var=code>

Este documento está protegido bajo la licencia de documentación libre *Free Documentacion License* del Proyecto GNU, para consulta ver el sitio <http://www.gnu.org/licenses/fdl.txt> , toda persona que lo desee está autorizada a usar, copiar y modificar este documento según los puntos establecidos en la «Licencia FDL»

