

# Snotra Tech Adonet Oracle Data Components Tutorial

© Snotra Tech 2006-2007

Author: Michael Milonov

Snotra Tech Adonet Oracle Data Components for .NET provides dataset which is intended for Oracle database. Components extend standard functions and features of ADO.NET components and provide the possibility to use pessimistic and optimistic concurrent models; refresh and lock single dataset's record, get server generated values, utilize partial data selection. Components based on Microsoft .NET Managed Provider for Oracle and take .NET 2.0.

Snotra Tech Adonet Oracle Data Components provide two major components: STAdonetOracleConnection and STAdonetOraDataTable.

*STAdonetOracleConnection* is used to connect to Oracle database. It's simply a wrapper for System.Data.OracleClient.OracleConnection class. Instead STAdonetOracleConnection on could use System.Data.OracleClient.OracleConnection.

*STAdonetOraDataTable* is main component, it represent Snotra Tech DataSet. It works like DataTable from standard .NET 2.0 distributions, but in contract the one STAdonetOraDataTable uses connected model and implements extra features like concurrent model selection, current row refreshing, partial data selection, immediate database constraints checking, errors handling and so on.

This tutorial contains the number of examples and explanations those, as I hope, allow you to understand main features of Snotra Tech Adonet Oracle Data Components and start to work with them easy.

If you have any questions or remarks concerning this tutorial please feel free to send e-mail at <mailto:info@snotratech.com> mailbox.

## Contents

|  |    |
|--|----|
| Prerequisites .....  | 2  |
| Installation of Snotra Tech Adonet Oracle Data Components. ....        | 2  |
| First application .....  | 2  |
| Problem to solve .....   | 2  |
| Creating a Sample data table .....                                     | 2  |
| Creating application.....  | 3  |
| Making STAdonetOracleDataTable to be "ReadOnly" .....                  | 5  |
| Using automatic row refreshing .....                                   | 6  |
| Using complex query in Snotra Tech Adonet Oracle Data Components ..... | 7  |
| Problem to solve .....   | 8  |
| Creating sample data table.....  | 8  |
| Creating application.....  | 8  |
| Adding and deleting rows.....  | 9  |
| Refreshing current data row manually.....                              | 9  |
| Refreshing all records .....   | 10 |
| Customising columns of STAdonetOracleDataTable .....                   | 10 |
| The selection of concurrent model .....                                | 12 |
| Errors handling .....  | 13 |
| Partial data selection .....   | 15 |
| Creating master-detail application.....                                | 15 |
| Problem to solve .....   | 15 |

|   |    |
|---|----|
| Creating Sample data tables .....   | 15 |
| Creating application.....   | 16 |
| Editing data using master-detail application .....                            | 20 |
| Customizing columns collection of STAdonetOraDataTable .....                  | 22 |
| Using PL/SQL functions, ref cursors and ApplyRecord event .....               | 23 |
| Problem to solve .....  | 24 |
| Creating application.....   | 24 |
| In brief .....  | 26 |
| Creating lookup fields using Snotra Tech Adonet Oracle Data Components .....  | 26 |
| Problem to solve .....  | 26 |
| Creating application.....   | 26 |
| Using Snotra Tech Adonet Oracle Data Components for console applications..... | 30 |
| Problem to solve .....  | 30 |
| Creating application.....   | 30 |

## Prerequisites

.NET 2.0

Oracle Client 8i or higher.

Microsoft Visual Studio 2005.

Snotra Tech Adonet Oracle Data Components. Can be downloaded from here:

[http://www.snotratech.com/products/snocnet\\_adonet\\_demo.exe](http://www.snotratech.com/products/snocnet_adonet_demo.exe)

*Remark:*

*Microsoft .NET Managed Provider for Oracle is a part of .NET 2.0 distribution, so no additional software required.*

## Installation of Snotra Tech Adonet Oracle Data Components.

Run installer snocnet\_adonet.exe or snocnet\_adonet\_demo.exe and simply follow to instructions on the screen.

After successful installation you will see a new folder in your C:\Program Files catalog (or in custom place if you have changed folder during installation).

The folder *Snotra Tech/SNOC.NET.ADONET/bin* contains Snotra.Data.OracleClient.dll;

*Snotra Tech/SNOC.NET.ADONET/Documentation* contains developer's guide and this tutorial.

*Snotra Tech/SNOC.NET.ADONET/Samples* contains all samples those we discuss below (C# and Visual Basic versions).

## First application

Our tutorial we will start from simple .NET application that can work with Oracle database using Snotra Tech Adonet Oracle Data Components.

### ***Problem to solve***

Suppose we want to create a simple .NET application that has DataGridView on form and can get data from Oracle database table as well as edit, insert and delete data.

### ***Creating a Sample data table***

First we should make a sample table in Oracle database. We will use schema "scott" in our application but one can use any other. To create sample table "COLORS" run sqlplus.exe and execute following commands:

```
create table COLORS  
(
```

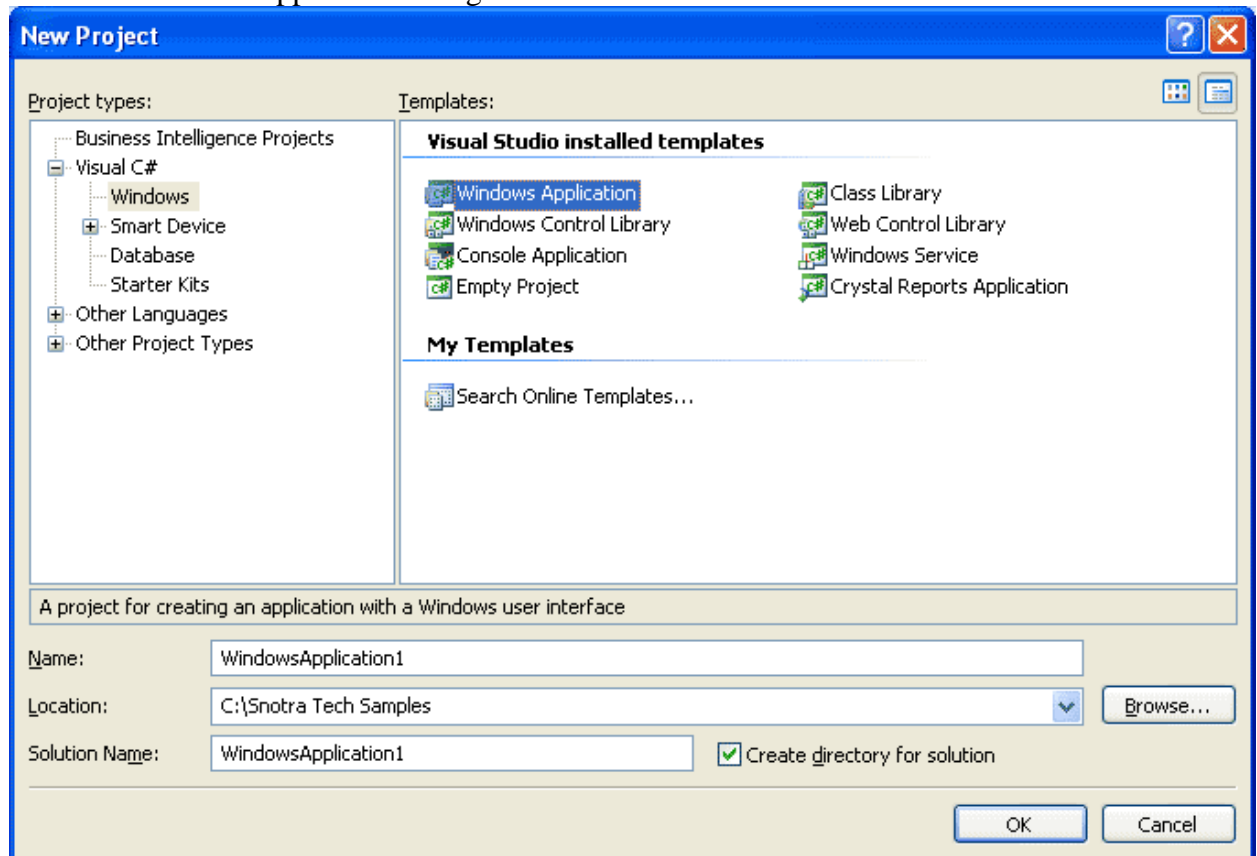
```
COLOR_ID NUMBER not null,  
COLOR_NAME varchar2(100) not null,  
RED number not null,  
GREEN number not null,  
BLUE number not null,  
CONSTRAINT PK_COLORS PRIMARY KEY (COLOR_ID)  
);  
--fill table-----  
insert into colors(color_id, color_name, red, green, blue) values(1, 'black', 0, 0, 0);  
insert into colors(color_id, color_name, red, green, blue) values(2, 'white', 254, 254, 254);  
insert into colors(color_id, color_name, red, green, blue) values(3, 'red', 254, 0, 0);  
insert into colors(color_id, color_name, red, green, blue) values(4, 'green', 0, 254, 0);  
insert into colors(color_id, color_name, red, green, blue) values(5, 'blue', 0, 0, 254);  
insert into colors(color_id, color_name, red, green, blue) values(6, 'yellow', 0, 254, 254);  
commit;
```

To make it easy create a text file, for example colors.txt, copy and paste the text above, save file and then execute following command:

```
sqlplus scott/tiger@ORCL @colors.txt.
```

## Creating application

Create a Windows Application using Visual Studio Wizard:



Then drag and drop three components to form: STAdonetOracleConnection, STAdonetOraDataTable and DataGridView.

First set ConnectionString property of STAdonetOracleConnection component:

**ConnectionString:** UserId=scott;Password=tiger;Data Source=orcl

Then set following properties of STAdonetOraDataTable component:

**BindingControl**=dataGridView1

**SQL**=select t.rowid, t.\* from colors t

**STAdonetOracleConnection**=stAdonetOracleConnection

*Binding Control property* is necessary for STAdonetOraDataTable to control DataGridView and return DataGridView's cursor when error or constraint's violation is raised during insert, delete or update commands. Also it is necessary to trace current data row in DataGridView.

*SQL property* is necessary to allow STAdonetOraDataTable to select appropriate data from Oracle database. Please notice that current SQL property contains ROWID. ROWID is essential to make STAdonetOraDataTable updatable. Without ROWID in SQL expression STAdonetOraDataTable will be read only.

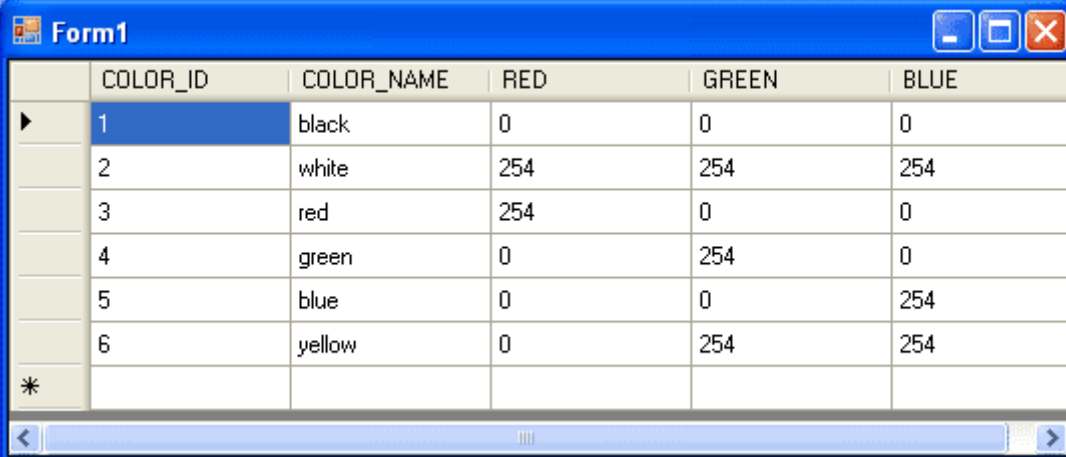
*STAdonetOracleConnection property* is necessary to allow STAdonetOraDataTable to connect to Oracle database.

Then open program code and write following text after InitializeComponent():

```
dataGridView1.DataSource = stOraDataTable1;  
stOracleConnection1.Open();  
stOraDataTable1.Open();
```

```
namespace WindowsApplication1  
{  
    public partial class Form1 : Form  
    {  
        public Form1()  
        {  
            InitializeComponent();  
            dataGridView1.DataSource = stOraDataTable1;  
            stOracleConnection1.Open();  
            stOraDataTable1.Open();  
        }  
    }  
}
```

Compile and run program, you should see following window:

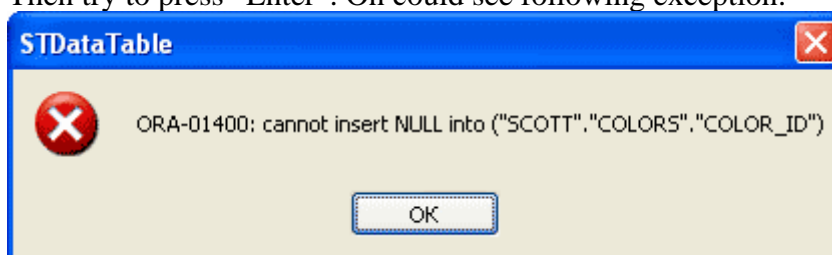


|   | COLOR_ID | COLOR_NAME | RED | GREEN | BLUE |
|---|----------|------------|-----|-------|------|
| ▶ | 1        | black      | 0   | 0     | 0    |
|   | 2        | white      | 254 | 254   | 254  |
|   | 3        | red        | 254 | 0     | 0    |
|   | 4        | green      | 0   | 254   | 0    |
|   | 5        | blue       | 0   | 0     | 254  |
|   | 6        | yellow     | 0   | 254   | 254  |
| * |          |            |     |       |      |

With three lines of code you have got a fully functional program that can show data from database as well as insert update and delete selected data. You can add new color, for example you want to add color named “my color” like in following figure:

|   | COLOR_ID | COLOR_NAME | RED | GREEN | BLUE |
|---|----------|------------|-----|-------|------|
|   | 1        | black      | 0   | 0     | 0    |
|   | 2        | white      | 254 | 254   | 254  |
|   | 3        | red        | 254 | 0     | 0    |
|   | 4        | green      | 0   | 254   | 0    |
|   | 5        | blue       | 0   | 0     | 254  |
|   | 6        | yellow     | 0   | 254   | 254  |
| ✎ |          | my color   | 1   | 1     | 1    |
| * |          |            |     |       |      |

Then try to press “Enter”. On could see following exception:



Then press OK, you should see that current row is still on “my color” row, so on need to correct this constraint violation by typing some COLOR\_ID, for example 7. Then press “Enter” again and you could see a new color that is added to database.

|    | COLOR_ID | COLOR_NAME | RED | GREEN | BLUE |
|----|----------|------------|-----|-------|------|
|    | 1        | black      | 0   | 0     | 0    |
|    | 2        | white      | 254 | 254   | 254  |
|    | 3        | red        | 254 | 0     | 0    |
|    | 4        | green      | 0   | 254   | 0    |
|    | 5        | blue       | 0   | 0     | 254  |
|    | 6        | yellow     | 0   | 254   | 254  |
|    | 7        | my color   | 1   | 1     | 1    |
| ▶* |          |            |     |       |      |

The same way works for update and delete commands, STAdonetOracleDataTable checks all constraints “on the fly” and keeps DataGridView current row on appropriate position until error, raised by database constraint violation, is not corrected.

## Making STAdonetOracleDataTable to be “ReadOnly”

Simply remove “rowid” from SQL property and STAdonetOracleDataTable became is “ReadOnly”.

For example set SQL property to:



In our concrete case obviously we should enable **AfterInsert** sub property, because our trigger changes COLOR\_ID after insert. But strongly recommended to allow **AfterUpdate** sub property also, because if you decide to add another trigger that will change data during update process, you will not happen to recompile application.

So enable AfterInsert and AfterUpdate sub properties, recompile and run application. Try to insert new color again, for example:

COLOR\_NAME:my\_color2, RED:3, GREEN:2, BLUE:1. Press Enter and you should see that COLOR\_ID is not empty. It has number that corresponds to the number in COLOR table.

If values in database could be changed by triggers (after or before update or insert) it can be handled simply by allowing AfterInsert and AfterUpdate sub properties, but sometimes value in database table could be changed by another user or some server-side process, for example by job and it can happen occasionally. So when user have changed some data in client application and tries to post changes he can get concurrent violation error message "COLUMN is changed by another user". We can model this situation. First start two instances of program. Change some color using the first instance, for example COLOR\_NAME:black, RED:10, GREEN:10, BLUE:10. Then try to change the same color using second program instance, for example COLOR\_NAME:black, RED:1. Error message will be raised. How to cope with this situation? There are some ways:

1. On can refresh current row manually and retry to change data. How to refresh current row you can see in chapter **Refreshing current data row manually**.
2. On can select suitable concurrent model and lock record before edit. How to do so see the chapter **The selection of concurrent model**.
3. Allow **BeforeEdit** sub property and row will be re-fetched automatically when user have edited one column and go to another.

Let's try to allow **BeforeEdit** property. Close both applications, go to Visual Studio and set **BeforeEdit** to true. Change some color using the first instance, for example COLOR\_NAME:black, RED:0, GREEN:0, BLUE:0. Then try to change the same color using second program instance, for example COLOR\_NAME:black, RED:5. Then go to GREEN column using key "Tab" or mouse. You can notice that GREEN and BLUE columns change their values to 0. So you can change GREEN and BLUE columns values for example to 5 and post changes successfully.

## Using complex query in Snotra Tech Adonet Oracle Data Components

Usually we use complex SQL expressions those contain more then one table and some conditions (>, <, like, and etc).

Let me explain the inner refreshing mechanism of STAdonetOracleDataTable more explicitly. When a record is refreshed, by default only the fields belonging to the updating table will be fetched from the database. Derived fields that result from joins, function calls, calculations, and so on will therefore not be refreshed. If you want to refresh derived fields as well, enable the **RefreshOptions:AllFields** option.

This is accomplished by re-executing the SQL statement for just the current record. This requires that the where clause is extended with the rowid of the current record. Let's assume the following SQL statement:

```
select f.rowid, f.*, c.color_name from flowers f, colors c where
c.color_id=f.color_id
```

Before editing or after inserting or updating a record, the following statement will be executed to refresh all fields:

```
select f.rowid, f.*, c.color_name from flowers f, colors c where
c.color_id=f.color_id and f.rowid = :snotra__rowid
```



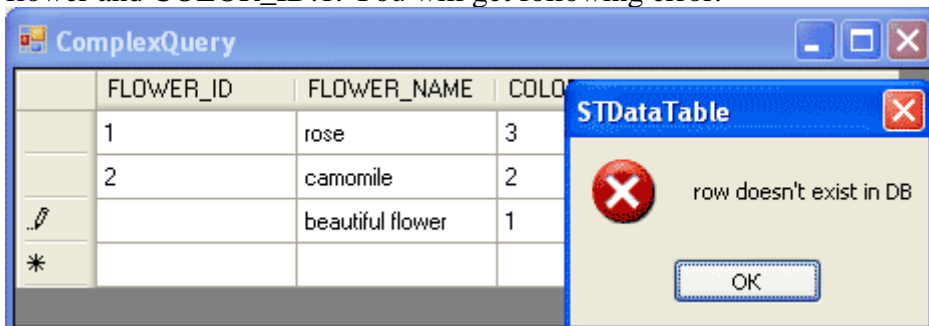
```
public ComplexQuery()
{
    InitializeComponent();
    try
    {
        stConnection.Open();
    }
    catch (Exception e)
    {
        MessageBox.Show(e.Message, "Error");
    }

    stAdonetOraDataTable1.BindingControl = dGrid;
    dGrid.DataSource = stAdonetOraDataTable1;

    stAdonetOraDataTable1.SQL =
        "select f.rowid, f.*, c.color_name from flowers f, " +
        "colors c where " +
        " c.color_id=f.color_id and f.flower_id < 3";

    stAdonetOraDataTable1.Open();
}
```

Start application and try to add a new flower, for example with FLOWER\_NAME: beautiful flower and COLOR\_ID:1. You will get following error:



Close application and change stAdonetOraDataTable1.SQL like:

```
stAdonetOraDataTable1.SQL =
    "select f.rowid, f.*, c.color_name from flowers f, colors c where " +
    " c.color_id=f.color_id /* END_REFRESH */ and f.flower_id < 3";
```

Try to do the same. Operation should be completed successfully.

## Adding and deleting rows

Return to our first application.

One can do it manually from DataGridView. But one can write a few lines of code also. Add two buttons on form: “Add record” and “Delete current”. Also add a code below:

```
private void Addbutton_Click(object sender, EventArgs e)
{
    STDataRow row = stAdonetOraDataTable1.AddNew();
    row["color_name"] = "MyColor";
    row["red"] = 20;
    row["green"] = 30;
    row["blue"] = 40;
    stAdonetOraDataTable1.Post();
}

private void DelButton_Click(object sender, EventArgs e)
{
    stAdonetOraDataTable1.Delete(stAdonetOraDataTable1.CurrentRecord);
}
```

## Refreshing current data row manually

Add a new button to form and name it “Refresh current”. Add following code to Click handler:

```
private void button1_Click(object sender, EventArgs e)
{
    stAdonetOraDataTable.RefreshRecord();
}
```

Start two instances of program. Change some color using the first instance, for example COLOR\_NAME:black, RED:10, GREEN:10, BLUE:10. Then position cursor on the same color in another program instance and push “Refresh current”. Then go to another row. Values in RED, GREEN, BLUE columns should be refreshed.

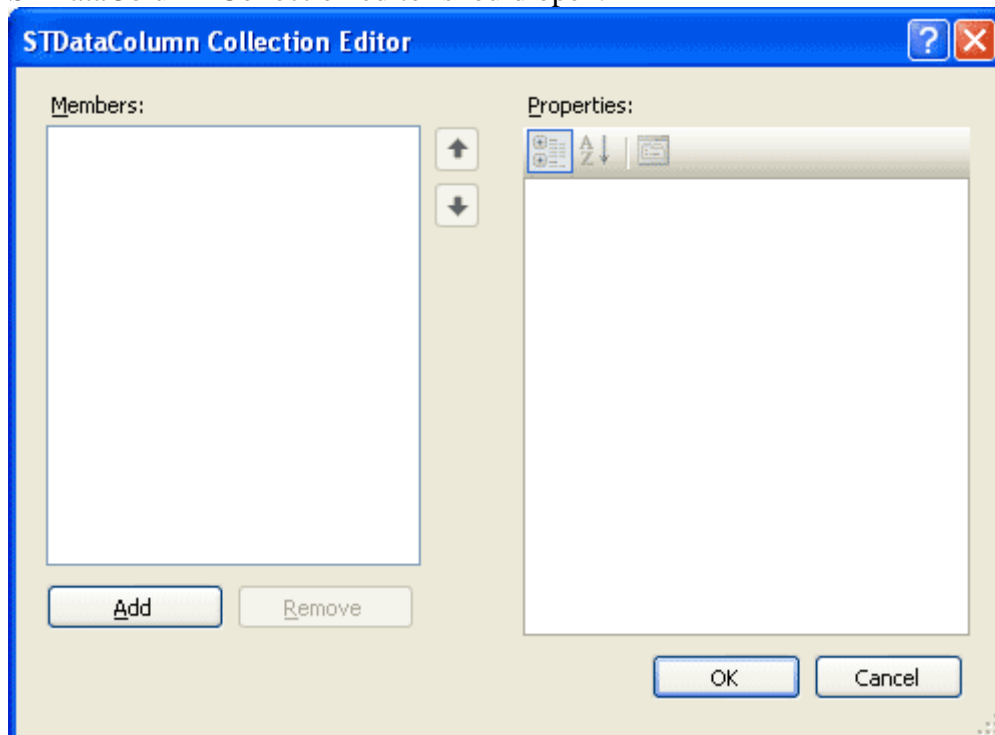
## Refreshing all records

To refresh all records in STAdonetOraDataTable add the following line of code:  
`stAdonetOraDataTable.Refresh();`

## Customising columns of STAdonetOracleDataTable

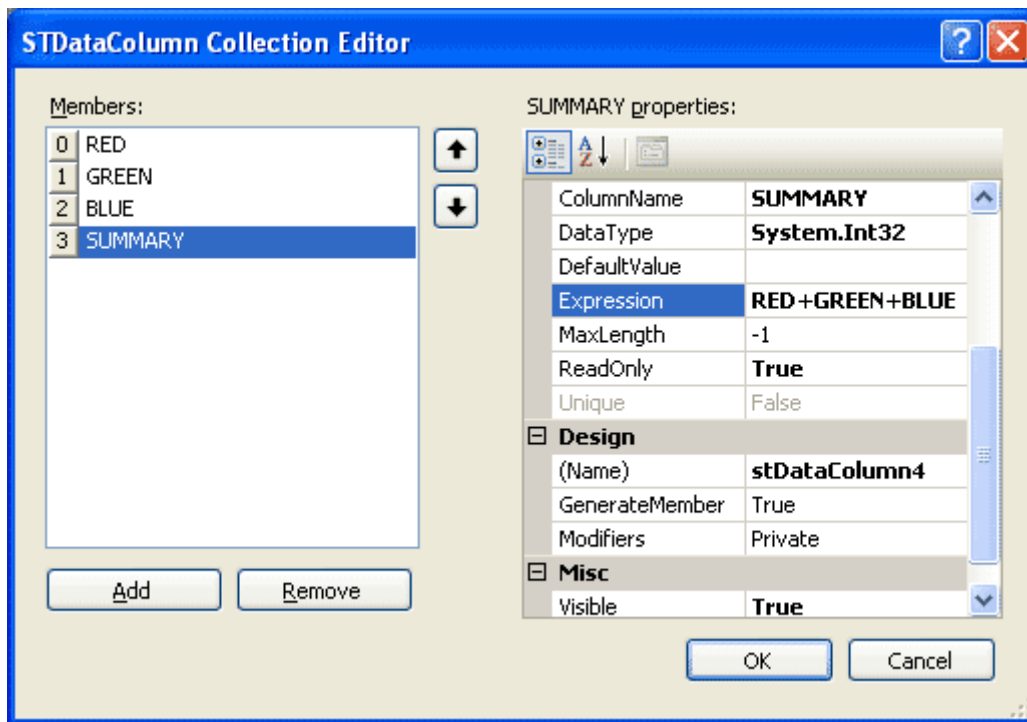
In our example we have used automatic columns creation. But one can to make custom columns and as well, as create calculated columns.

Suppose we want to show only four columns in dataGridView. We can do it by editing columns collection of stAdonetOraDataTable. Edit **Columns** property of stAdonetOraDataTable, STDataColumn Collection editor should open:

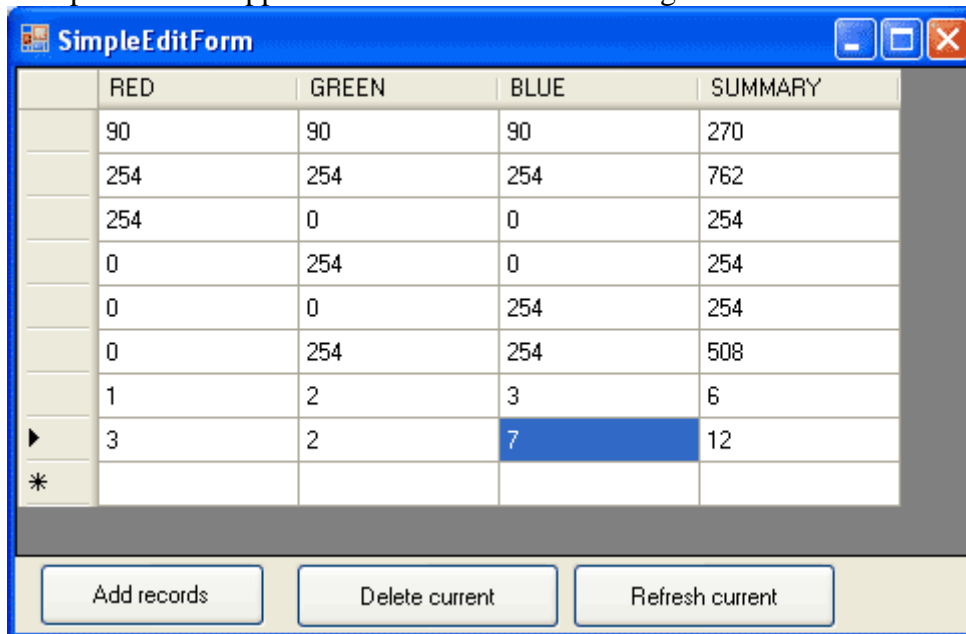


Suppose, we want to show columns RED, GREEN, BLUE and SUMMARY that is the sum of previous three columns.

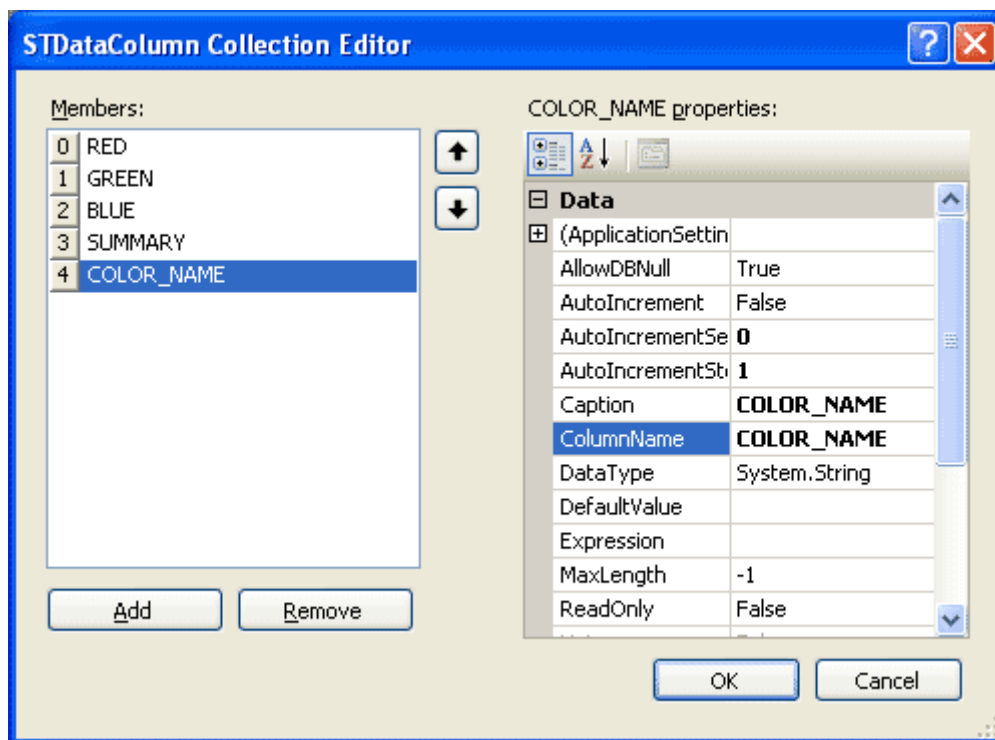
Add columns RED, GREEN, BLUE with DataType System.Int32 and column SUMMARY with DataType System.Int32 and Expression RED+GREEN+BLUE:



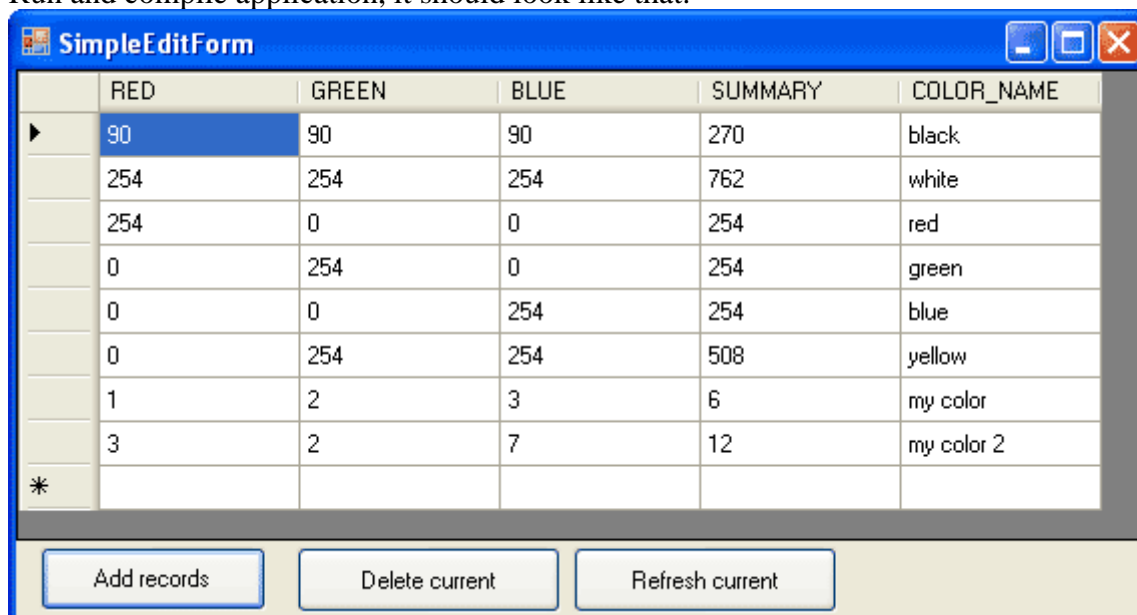
Compile and run application. We can see something like that:



Oh, we forgot about COLOR\_NAME, so go and add this column with DataType: System.String.



Run and compile application, it should look like that:



## The selection of concurrent model

Standard ADO.NET DataTable has only one concurrent model, it use the optimistic one. STAdonetOraDataTable can use optimistic as well as pessimistic concurrent models. You can select the suitable one using the property **LockingMode**.

LockingMode has four variants:

1. CheckImmediate - when the user starts editing a record, it is locked and a check is performed to see if it has been changed. The lock remains until the user posts or cancels the changes.
2. LockDelayed - when the user starts editing a record, a check is performed to see if it has been changed, but the record is not locked. Therefore, when the user posts the record, it is locked and checked again.

3. LockImmediate - when the user posts an edited record, it is locked and a check is performed to see if it has been changed. After this, the lock is released.
4. None - no locking or checks are performed. This should only be used in single user applications.

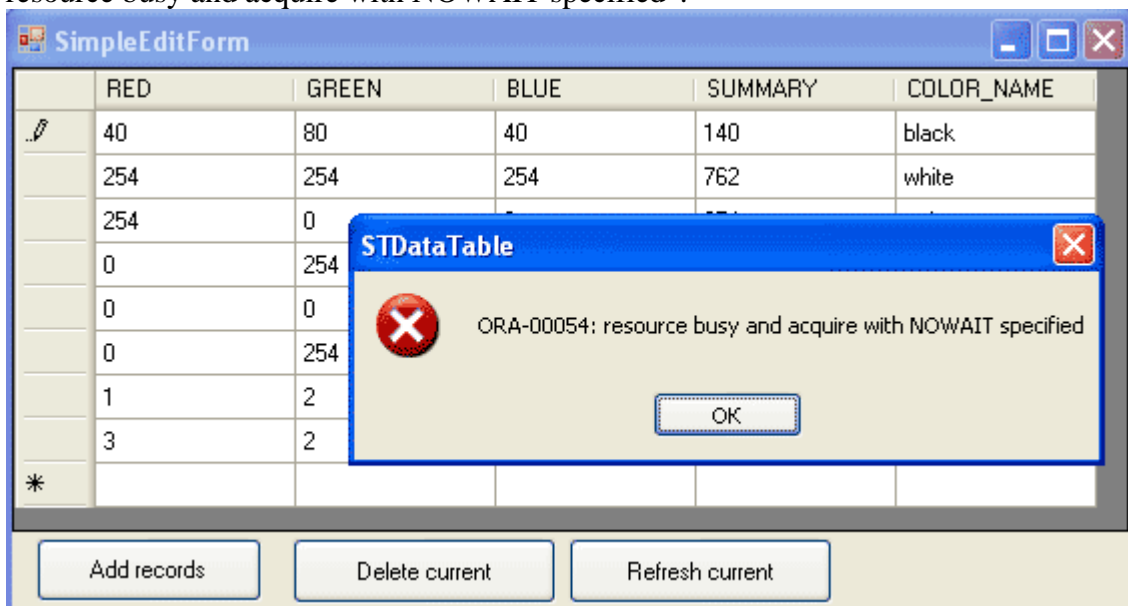
CheckImmediate is default Locking Mode. CheckImmediate and LockDelayed represent optimistic concurrent model. They lock record only immediately before posting changes. LockImmediate represent pessimistic concurrent model. It locks record at once when user have started to edit it. So if another user tries to change the same record he will get message that record is locked.

As for CheckImmediate see the chapter [Using automatic row refreshing](#).

STAdonetOraDataTable simply check record before post changes to database and if find that its record is not corresponds to record from database throws exception "record is changed by another user".

Let's try LockImmediate locking mode. Set LokingMode to LockImmediate recompile and run two instances of application.

In first instance change for example tow sells of color black: RED:10, BLUE:10. Don't post changes (don't changes row and don't press Enter), so the row with color "black" is editing now. The go to second instance of our program and try to change the same color, for example GREEN:50. Go to another cell or try to post changes. You will see an exception: "ORA-00054: resource busy and acquire with NOWAIT specified".



So this record have been locked by first instance of our program and it will be locked until you don't post changes. So go to first instance and post changes (press Enter or go to another row). The try to post changes from second program instance. You will see that changes will be posted successfully.

## Errors handling

STAdonetOraDataTable has default error handler. It simply intercepts Oracle exceptions and shows them in MessageBox. But one can to make custom error handler and show or hide exceptions according your wishes.

To replace default error handler use DataError message of STAdonetOraDataTable. For example:

```
private void stAdonetOraDataTable1_DataError(object sender, STDataErrorEventArgs e)
{
    MessageBox.Show(e.Exception.Message, "My corporation " + e.DataAction.ToString(),
        MessageBoxButtons.OK, MessageBoxIcon.Warning);
}
```

Then try to make any error, for example duplicate COLOR\_ID value... You will get error with custom look and feel.

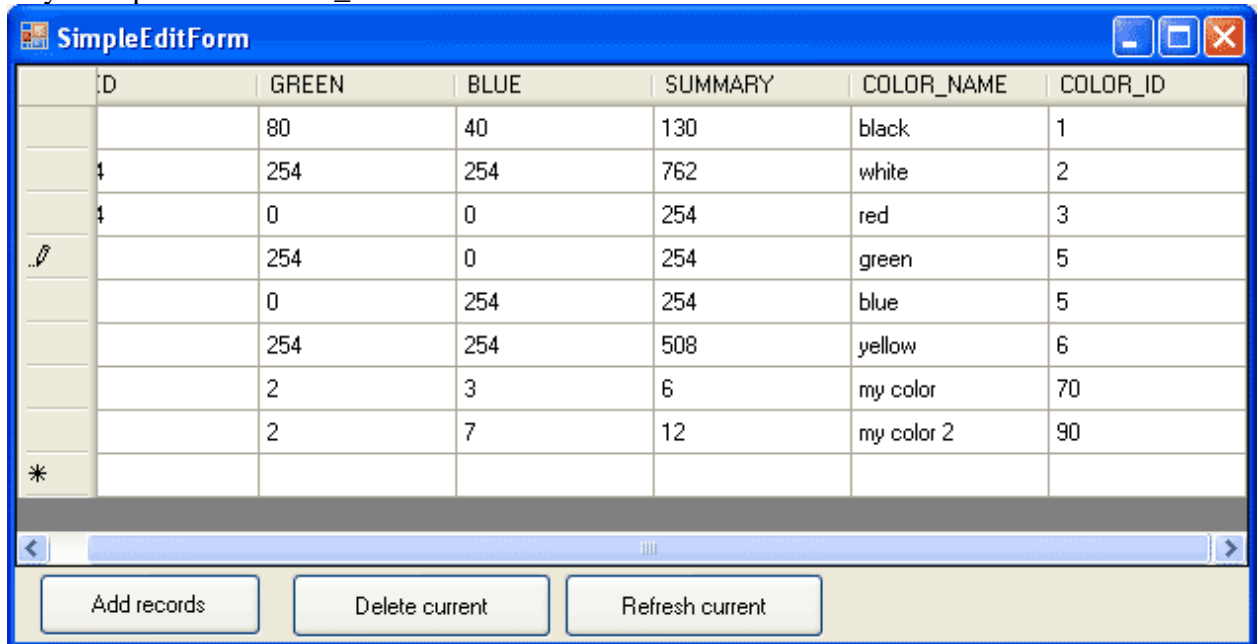
Also there are three other events that can help you:

- DeleteError – raises when exception occurred during delete operation;
- EditError – occurs when an application attempts to modify or insert a record and an exception is thrown;
- TranslateMessage - when an Oracle error occurs during an insert, update, delete or lock issued by the dataset, this event is triggered;

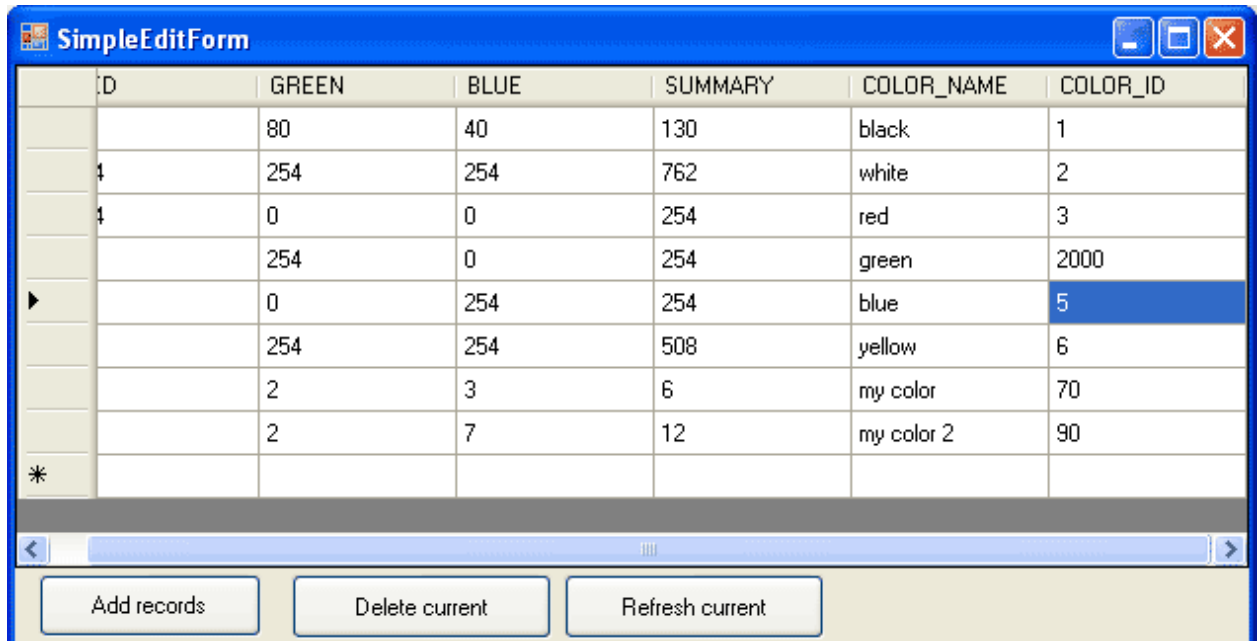
Using EditError and DeleteError one can for example retry Insert, Update or Delete action if it failed. To show it let's add EditError event and following code:

```
private void stAdonetOraDataTable1_EditError(object sender, STDataEventArgs e)
{
    e.Row["COLOR_ID"] = 2000;
    e.DataAction = STDataAction.Retry;
}
```

Try to duplicate COLOR\_ID column value:



You don't see error, because EditError event has caught it and change COLOR\_ID value from 5 to 2000:



Of course it can be more sophisticated algorithm, anyway on can to use `EditError` and `DeleteError` events to handle error messages and try to correct them “on the fly”.

`TranslateMessage` is informative event, because contains oracle error code, action name (Check, Lock, Insert, Update, Delete, Refresh) and error message text. One can to use it for enhanced errors monitoring.

## Partial data selection

Some queries can return lots of rows, hundreds and thousands rows. Obviously that `DataGridView` has limited size to show rows and will be better to get data rows from Oracle database according the position of scroll tab.

Suppose we have three hundreds rows in our `COLOR` table. Oh, stop we have only a few rows. Ok, let’s add them. Add a new button, named “300 rows” as well as following code:

```
private void button_300rows_Click(object sender, EventArgs e)
{
    for (int i = 0; i < 300; i++)
    {
        STDataRow row = stAdonetOraDataTable1.AddNew();
        row["color_name"] = "MyColor";
        row["red"] = 20;
        row["green"] = 30;
        row["blue"] = 40;
        stAdonetOraDataTable1.Post();
    }
}
```

Compile and run application, push “300 rows” button. Well, our table has a lot of rows now. Then close application and run it again. Scroll down data in `DataGridView`. You can notice that data are fetched from database according the scroll tab position. Convenient... isn’t it?

## Creating master-detail application

Usually ADO.NET programmers create master-detail relationship using `DataSet` component. Snotra Tech Adonet Oracle Data Components use their own mechanism to create master-detail application. You can set this kind of relationship directly in design-time using visual editor or programmatically in runtime.

### ***Problem to solve***

Suppose we have two tables in Oracle database with have master-detail relationship. We want to make .NET application that can get data from those two tables as well as edit, insert and delete data.

### ***Creating Sample data tables***

In our tutorial we will use two tables: `DEPT` and `EMP`:

```
CREATE TABLE DEPT (
    DEPTNO NUMBER(2) CONSTRAINT PK_DEPT PRIMARY KEY,
    DNAME VARCHAR2(14) ,
    LOC VARCHAR2(13)
)
/
CREATE TABLE EMP (
    EMPNO NUMBER(4) CONSTRAINT PK_EMP PRIMARY KEY,
    ENAME VARCHAR2(10),
    JOB VARCHAR2(9),
    HIREDATE DATE,
    DEPTNO NUMBER(2) CONSTRAINT FK_DEPTNO REFERENCES DEPT
)
/
INSERT INTO DEPT values (10,'ACCOUNTING','NEW YORK');
INSERT INTO DEPT VALUES (20,'RESEARCH','DALLAS');
INSERT INTO DEPT values (30,'SALES','CHICAGO');
INSERT INTO DEPT values (40,'OPERATIONS','BOSTON')
/
```

```
INSERT INTO EMP values (7369, 'SMITH', 'CLERK', to_date('17-12-1980', 'dd-mm-yyyy'), 20);
INSERT INTO EMP values (7499, 'ALLEN', 'SALESMAN', to_date('20-2-1981', 'dd-mm-yyyy'), 30);
INSERT INTO EMP values (7521, 'WARD', 'SALESMAN', to_date('22-2-1981', 'dd-mm-yyyy'), 30);
INSERT INTO EMP values (7566, 'JONES', 'MANAGER', to_date('2-4-1981', 'dd-mm-yyyy'), 20);
INSERT INTO EMP values (7654, 'MARTIN', 'SALESMAN', to_date('28-9-1981', 'dd-mm-yyyy'), 30);
INSERT INTO EMP values (7698, 'BLAKE', 'MANAGER', to_date('1-5-1981', 'dd-mm-yyyy'), 30);
INSERT INTO EMP values (7782, 'CLARK', 'MANAGER', to_date('9-6-1981', 'dd-mm-yyyy'), 10);
INSERT INTO EMP values (7788, 'SCOTT', 'ANALYST', to_date('13-07-1987', 'dd-mm-yyyy'), 20);
INSERT INTO EMP values (7839, 'KING', 'PRESIDENT', to_date('17-11-1981', 'dd-mm-yyyy'), 10);
INSERT INTO EMP values (7844, 'TURNER', 'SALESMAN', to_date('8-9-1981', 'dd-mm-yyyy'), 30);
INSERT INTO EMP values (7876, 'ADAMS', 'CLERK', to_date('13-07-1987', 'dd-mm-yyyy'), 20);
INSERT INTO EMP values (7900, 'JAMES', 'CLERK', to_date('3-12-1981', 'dd-mm-yyyy'), 30);
INSERT INTO EMP values (7902, 'FORD', 'ANALYST', to_date('3-12-1981', 'dd-mm-yyyy'), 20);
INSERT INTO EMP values (7934, 'MILLER', 'CLERK', to_date('23-1-1982', 'dd-mm-yyyy'), 10)
/
commit
/
```

To make it easy create a text file, for example master-det.txt, copy and paste the text above, save file and then execute following command:

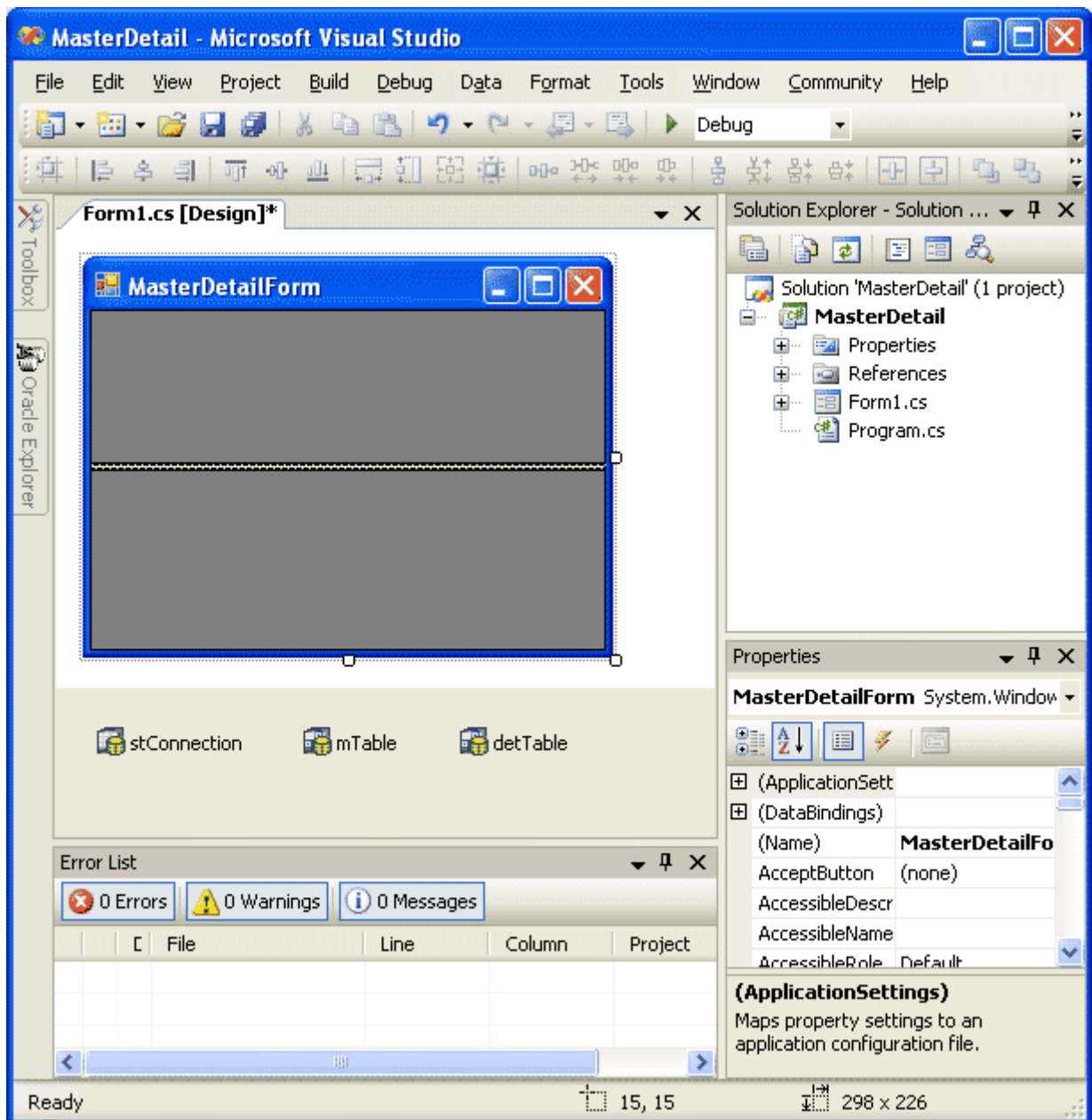
```
sqlplus scott/tiger@ORCL @master-det.txt
```

Then leave sqlplus by typing command “exit”. Now we are ready to make application using Microsoft Visual Studio 2005.

### **Creating application**

Create a Windows Application using Visual Studio Wizard, just like in chapter [First Application](#).

Then drop to form STAdonetOracleConnection and rename it as stConnection and two STAdonetOraDataTable components and rename them as mTable and detTable correspondingly. Also drop to form two DataGridViews and rename them as mGrid and detGrid. Also one can place splitter between them but it is optionally. After that you can see form as on figure below:



Then setConnectionString property of stConnection to

```
Data source=orcl;User Id=scott;Password=tiger
```

Or use your own Oracle database name, username and password.

Select mTable and set following properties:

**Binding Control** to mGrid,

**SQL** to select t.rowid, t.\* from dept t

**STOracleConnection** to stConnection

Then select detTable and set following properties:

**Binding Control** to detGrid,

**Detail Fields** to deptno

**Master** to mTable

**MasterFields** to deptno

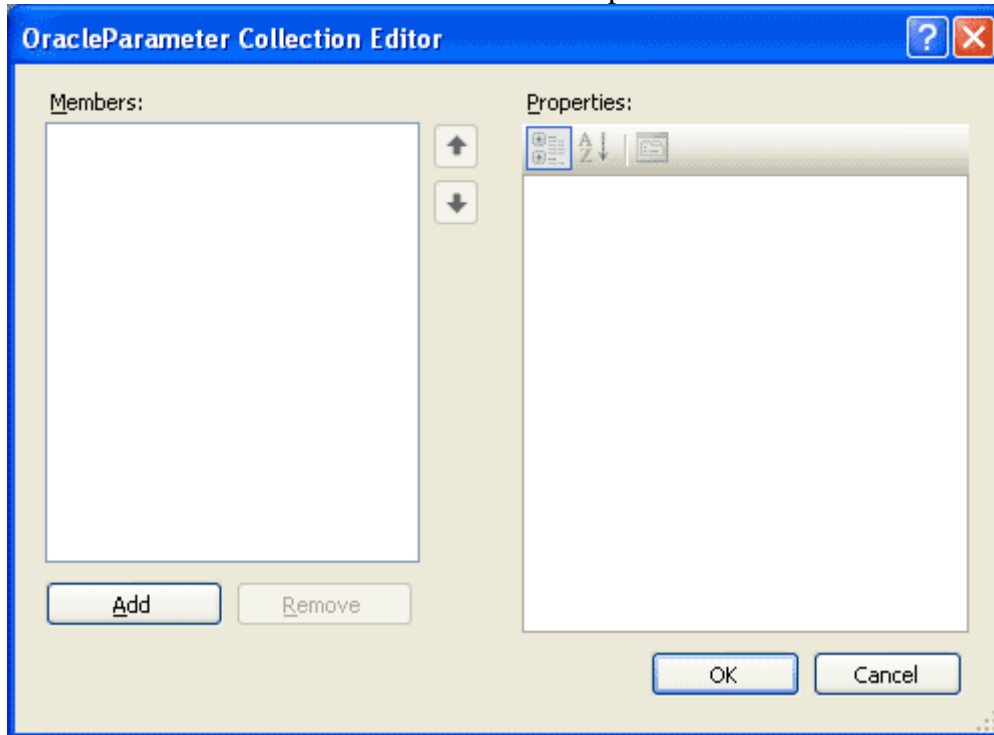
**SQL** to select t.rowid, t.\* from emp t where t.deptno = :deptno

**STOracleConnection** to stConnection

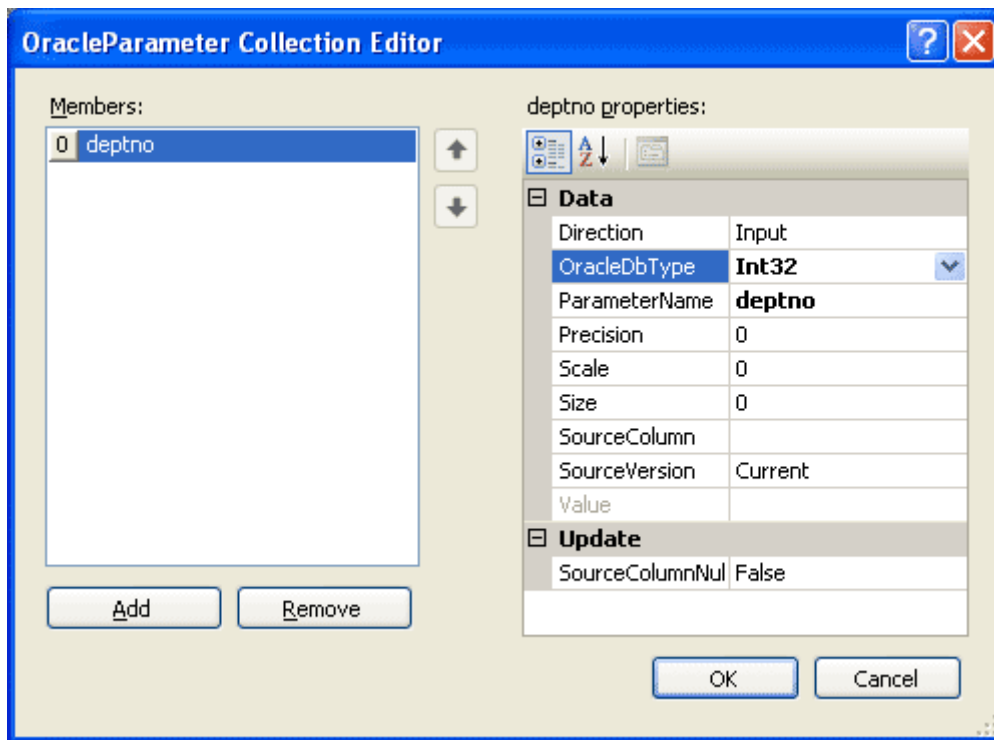
Tables DEPT and EMP have one to many relationship through column DEPTNO. mTable is master table for detTable, so we have set Master property to mTable. Also we have set MasterFields and DetailFields properties according to their name in corresponding tables in Oracle database. If our table has complex primary key, for example DEPTNO, DNAME we can set MasterFields to DEPTNO, DNAME and change DetailFields property correspondingly.

Pay your attention to SQL expression: **where t.deptno = :deptno**. **:deptno** is variable. It must be declared for detTable. When user browses mTable, SToraDataTable component sets appropriate **:deptno** value for detTable and refreshes it so edit **Variables** property of detTable.

OracleParameter Collection Editor should be opened:



Add a new variable using “Add” button as on figure below:



Set **ParameterName** to **deptno** and **OracleDbType** to **Int32** and push "OK". Great, now we should add a few lines on code, so open application code and add following text to form constructor after `InitializeComponent();`:

```
try
{
    stConnection.Open();
}
catch (Exception e)
{
    MessageBox.Show(e.Message, "Error");
}

mGrid.DataSource = mTable;
detGrid.DataSource = detTable;
mTable.Open();
detTable.Open();
```

Your application code should look like:

```

namespace MasterDetail
{
    public partial class MasterDetailForm : Form
    {
        public MasterDetailForm()
        {
            InitializeComponent();
            try
            {
                stConnection.Open();
            }
            catch (Exception e)
            {
                MessageBox.Show(e.Message, "Error");
            }

            mGrid.DataSource = mTable;
            detGrid.DataSource = detTable;
            mTable.Open();
            detTable.Open();
        }
    }
}

```

Compile and run application. The following window should appears

The screenshot shows a Windows application window titled "MasterDetailForm". It contains two data grids. The top grid has columns for deptno, dname, and loc. The bottom grid has columns for EMPNO, ENAME, JOB, HIREDATE, and DEPTNO. The application is running and displaying data from a database.

| deptno | dname      | loc      |
|--------|------------|----------|
| 10     | ACCOUNTING | NEW YORK |
| 20     | RESEARCH   | DALLAS   |
| 30     | SALES      | CHICAGO  |
| 40     | OPERATIONS | BOSTON   |
| *      |            |          |

| EMPNO | ENAME  | JOB       | HIREDATE   | DEPTNO |
|-------|--------|-----------|------------|--------|
| 7782  | CLARK  | MANAGER   | 09.06.1981 | 10     |
| 7839  | KING   | PRESIDENT | 17.11.1981 | 10     |
| 7934  | MILLER | CLERK     | 23.01.1982 | 10     |
| *     |        |           |            |        |

### ***Editing data using master-detail application***

Navigate mGrid, select another row and data in detGrid should be changed according deptno value.

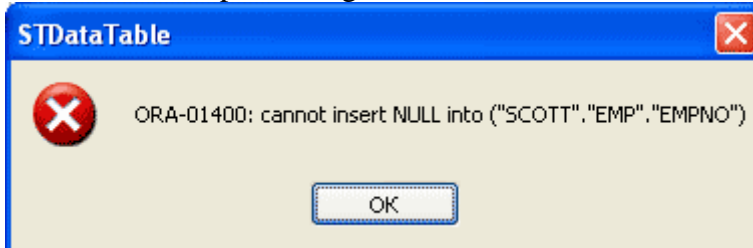
Then try to add new record in detGrid, for example like here:

| deptno | dname      | loc      |
|--------|------------|----------|
| 10     | ACCOUNTING | NEW YORK |
| 20     | RESEARCH   | DALLAS   |
| 30     | SALES      | CHICAGO  |
| 40     | OPERATIONS | BOSTON   |
| *      |            |          |

| EMPNO | ENAME  | JOB       | HIREDATE   | DEPTNO |
|-------|--------|-----------|------------|--------|
| 7782  | CLARK  | MANAGER   | 09.06.1981 | 10     |
| 7839  | KING   | PRESIDENT | 17.11.1981 | 10     |
| 7934  | MILLER | CLERK     | 23.01.1982 | 10     |
|       | mike   | CEO       |            | 10     |
| *     |        |           |            |        |

Press “Enter” to post changes to Oracle database. Following exception is raised:



STOraDataTable catches database exceptions “on the fly” and post data automatically when you press “Enter” or change current row. If exception is raised STOraDataTable returns focus to row that accounts for exception. Type any number in EMPNO column and retry to press “Enter”.

| deptno | dname      | loc      |
|--------|------------|----------|
| 10     | ACCOUNTING | NEW YORK |
| 20     | RESEARCH   | DALLAS   |
| 30     | SALES      | CHICAGO  |
| 40     | OPERATIONS | BOSTON   |
| *      |            |          |

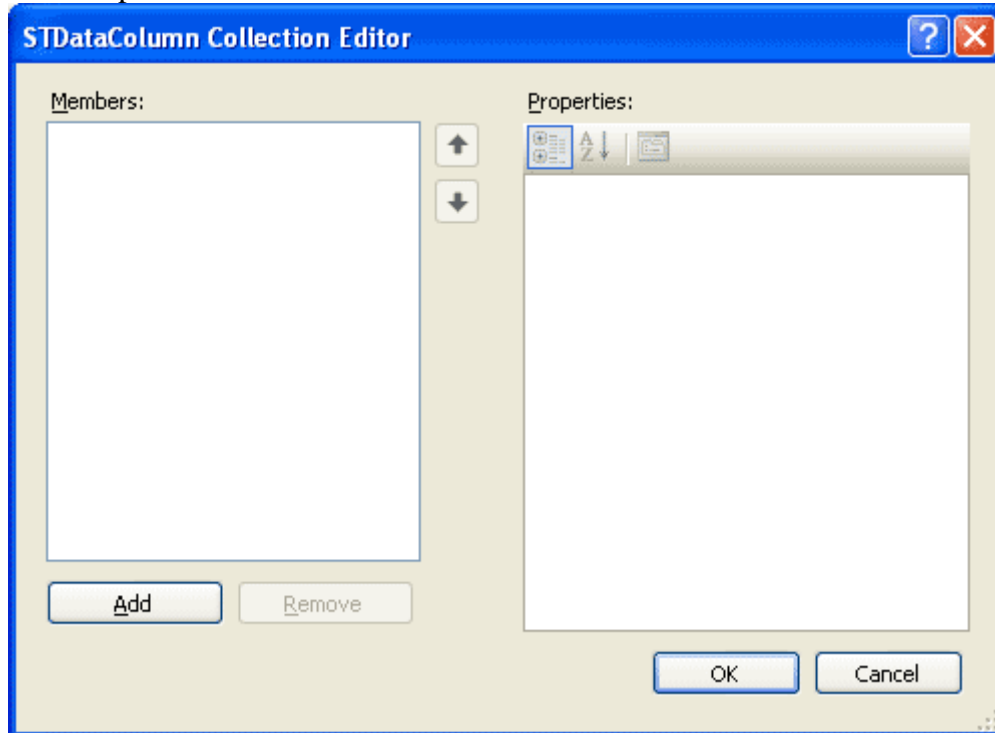
  

| EMPNO | ENAME  | JOB       | HIREDATE   | DEPTNO |
|-------|--------|-----------|------------|--------|
| 7782  | CLARK  | MANAGER   | 09.06.1981 | 10     |
| 7839  | KING   | PRESIDENT | 17.11.1981 | 10     |
| 7934  | MILLER | CLERK     | 23.01.1982 | 10     |
| 1111  | mike   | CEO       |            | 10     |
| ▶*    |        |           |            | 10     |

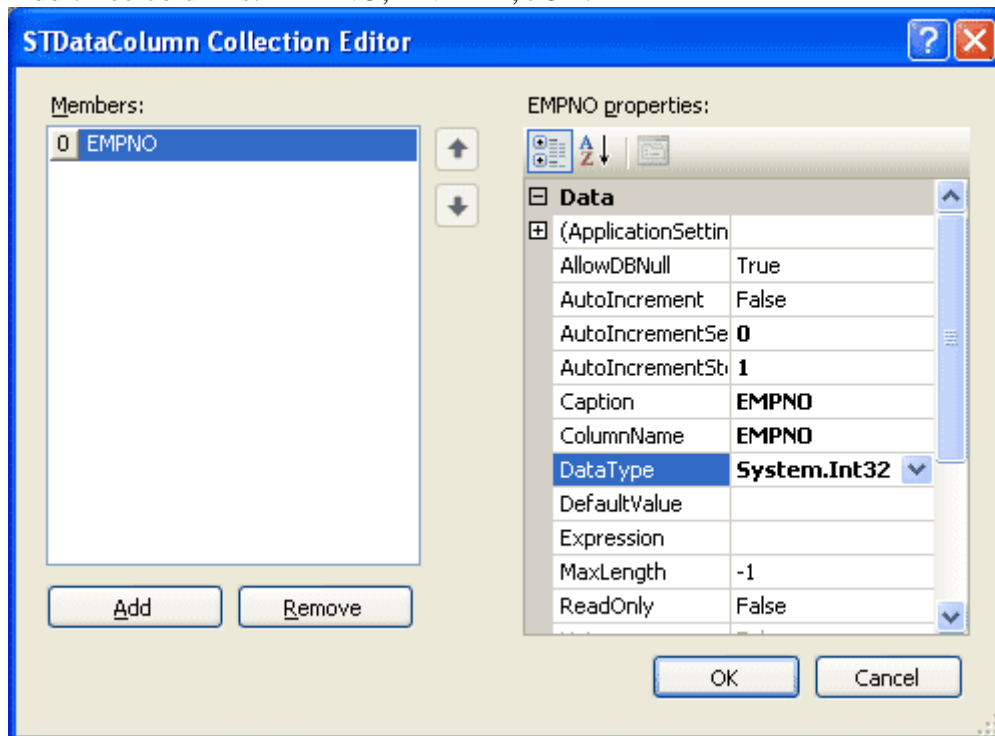
New row is added and posted to Oracle database. You can also modify or delete any row. If database exception is raised you will be informed at once.

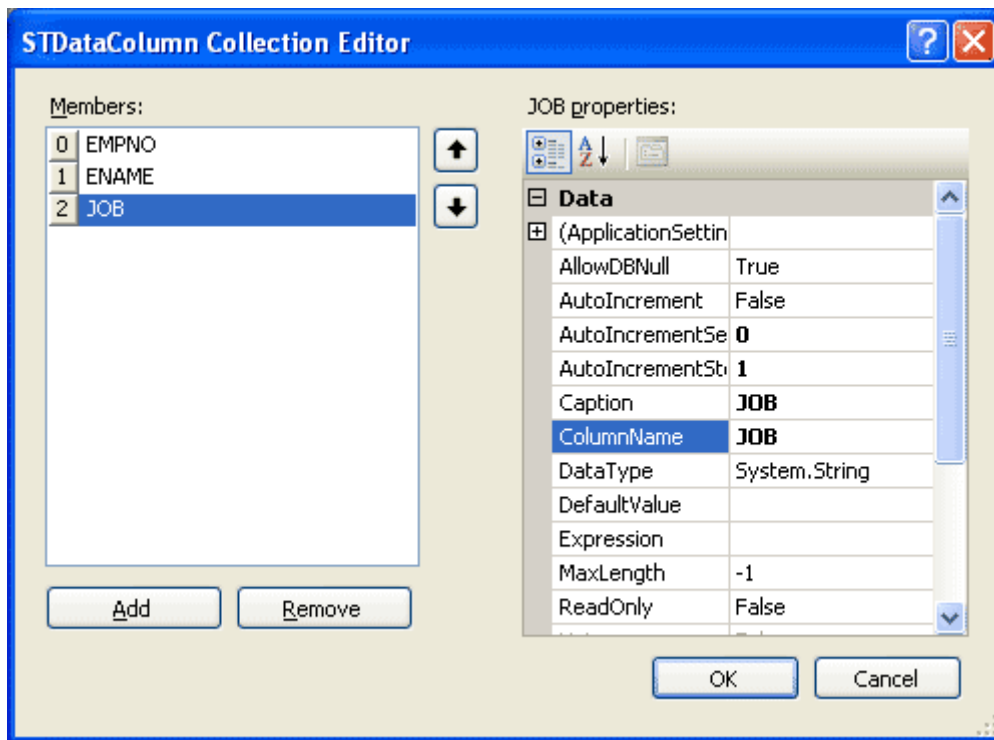
### Customizing columns collection of STAdonetOraDataTable

Suppose we want to show only three columns in detGrid. We can do it by editing columns collection of detTable. Edit **Columns** property of detTable, STDataColumn Collection editor should open:



Add three columns: EMPNO, ENAME, JOB:





Then compile and run application, you should see the following window:



## Using PL/SQL functions, ref cursors and ApplyRecord event

Snotra Tech Adonet Oracle Data Components use their own mechanism to access and edit data in Oracle database. It is suitable for the majority of programmer's tasks. But often to access and modify data developers use server-side PL/SQL functions. Our components support this possibility and provide convenient way to use PL/SQL functions to access and modify data in Oracle database.

## Problem to solve

Suppose we want to make .NET application that can get data from Oracle database as well as insert, delete and edit. Also we want to use PL/SQL function select data from COLORS table and insert data into COLORS table.

## Creating application

Create a .NET Windows Application just like in chapter [First Application](#). Add STAdonetOracleConnection, STAdonetOraDataTable, DataGridView on form and set component's properties just like in this chapter. We will use the same table COLORS so create it if you don't have the one and don't forget add trigger as in chapter [Using Automatic row refreshing](#).

Let's create a package FUNCTION\_USE that has two functions: GetColors and InsertColor:

create or replace package FUNCTION\_USE is

```
type colors_row is record(
    rowid varchar(50),
    color_id number,
    color_name varchar(100),
    red number,
    green number,
    blue number
);

TYPE colors_row_cursor IS REF CURSOR return colors_row;

function GetColors(red_v number) return colors_row_cursor;
function InsertColor(color_name_a varchar2, red_a number, green_a number,
    blue_a number) return varchar2;

end FUNCTION_USE;
/
create or replace package body FUNCTION_USE is

function GetColors(red_v number) return colors_row_cursor is

res_cursor colors_row_cursor ;
begin

open res_cursor for
    select c.rowid, c.color_id, c.color_name, c.red,
        c.green, c.blue from colors c where c.red > red_v;

return res_cursor;
end;

function InsertColor(color_name_a varchar2, red_a number, green_a number,
    blue_a number) return varchar2 is

ret_val varchar2(50);
begin
insert into colors (color_name, red, green, blue)
    values(color_name_a, red_a, green_a, blue_a) returning rowid into ret_val;
return ret_val;
end;

end FUNCTION_USE;
```

The function GetColors returns ref cursor of colors\_row. Pay your attention that type "colors\_row" contains field named rowid. Rowid is necessary to allow STAdonetOraDataTable to be editable. Without rowid field it will be read only (see [Making STAdonetOraDataTable to be "Read Only"](#) chapter). GetColors also takes one input argument (red\_v). It select all rows from COLORS table where red column value is bigger then red\_v.

So we have one input variable and one output (ref cursor). Let's add output and input variables to STAdonetOraDataTable in form's constructor:

```
public FunctionsUse()
{
    InitializeComponent();
    try
```

```

    {
        stConnection.Open();
    }
    catch (Exception e)
    {
        MessageBox.Show(e.Message, "Error");
    }
    dGrid.DataSource = stAdonetOraDataTable1;
    stAdonetOraDataTable1.BindingControl = dGrid;

    //function return variable should be added first!!!
    stAdonetOraDataTable1.DeclareVariable("result",
        System.Data.OracleClient.OracleType.Cursor,
        ParameterDirection.ReturnValue);
    //declare input variable
    stAdonetOraDataTable1.DeclareVariable("red_v",
        System.Data.OracleClient.OracleType.Int32);
    //select all colors where red > 10
    stAdonetOraDataTable1.SetVariable("red_v", 10);
    stAdonetOraDataTable1.Open();
}

```

Let's set **SQL** property of **STAdonetOraDataTable** to **FUNCTION\_USE.GetColors**, **SQLCommandType** to **StoredProcedure** and **UpdatingTable** to **COLORS**.

**UpdatingTable** is very important property here, because it is used by **STAdonetOraDataTable** to generate SQL expressions to check, lock, update, insert, delete and refresh record. But why we didn't use this property early? We have used SQL expressions like "select t.rowid, t.\* from colors" and **STAdonetOraDataTable** by default gets the first table name from **SQL** property to generate its own SQL expressions. Now we use PL/SQL function to make select from database and **UpdatingTable** property should be assigned manually. Also we will replace insert operation, but SQL expressions to check, lock, update, refresh and delete will be generated automatically by **STAdonetOraDataTable** so **UpdatingTable** property is actual here. All operations those have been mentioned here (check, lock, update, insert, delete and refresh) could be replaced by .NET developer. **STAdonetOraDataTable** has **ApplyRecord** event that allows customizing them.

Let's customize insert operation. Make **ApplyRecord** event handler for **STAdonetOraDataTable** and put following code:

```

private void stAdonetOraDataTable1_ApplyRecord(
    Snotra.Data.OracleClient.STAdonetOraDataTable sender,
    Snotra.Data.STApplyRecordEventArgs e)
{
    OracleCommand oracleDMLCommand = null;
    if (e.Action == Snotra.Data.STDBAction.Insert)
    {
        try
        {
            oracleDMLCommand = stConnection.CreateCommand();
            oracleDMLCommand.CommandType = CommandType.StoredProcedure;
            oracleDMLCommand.CommandText = "function_use.insertcolor";
            OracleParameter param =
                oracleDMLCommand.Parameters.Add("ret_value",
                    OracleType.VarChar, 50);
            param.Direction = ParameterDirection.ReturnValue;
            //-----
            oracleDMLCommand.Parameters.Add("color_name_a",
                OracleType.VarChar).Value =
                e.Row["color_name"];
            oracleDMLCommand.Parameters.Add("red_a", OracleType.Int32).Value =
                e.Row["red"];
            oracleDMLCommand.Parameters.Add("green_a", OracleType.Int32).Value =
                e.Row["green"];
            oracleDMLCommand.Parameters.Add("blue_a", OracleType.Int32).Value =
                e.Row["blue"];
            //-----
            oracleDMLCommand.ExecuteNonQuery();
            //set returned new rowid
            e.NewRowId =
                Convert.ToString(oracleDMLCommand.Parameters["ret_value"].Value);
            //not execute default insert, apply current
            e.Apply = true;
        }
    }
}

```

```
        catch (Exception ex)
        {
            //throw exception if any
            throw ex;
        }
        finally
        {
            if (oracleDMLCommand != null)
                oracleDMLCommand.Dispose();
        }
    }
}
```

We simply insert the row manually, set `NewRowId` returned by `OracleCommand` and set `Snotra.Data.STApplyRecordEventArgs.Apply` to true, to let to `STAdonetOraDataTable` know that we have used customized operation.

### ***In brief***

If you want to get data using PL/SQL function on need:

- PL/SQL function should return a ref cursor;
- Ref cursor should contain a ROWID field;
- Set SQL property to PL/SQL function name;
- Set `SQLCommandType` property to `StoredProcedure`;
- Set `UpdatingTable` property;
- Add output and input variables to `STAdonetOraDataTable`.

If you want to customize check, lock, update, insert, delete or refresh operations on need:

- Use `ApplyRecord` event;
- Set `Snotra.Data.STApplyRecordEventArgs.NewRowId` in case of insert command;
- Set `Snotra.Data.STApplyRecordEventArgs.Apply` to true;

## **Creating lookup fields using Snotra Tech Adonet Oracle Data Components**

Lookup fields are used widely in visual applications. Simple example first. We have dictionary table `WORKERS` in our database that contains worker names of our firm. Each worker has first name, last name and of course he has unique `WORKER_ID` that we use as primary key. We use `WORKER_ID` in another tables linking to `WORKERS` table. But when we build interface of our .NET application we want to show last name of first name but not `WORKER_ID`. So we should use lookup fields that show some string value instead ID and allow selecting some worker from workers list.

### ***Problem to solve***

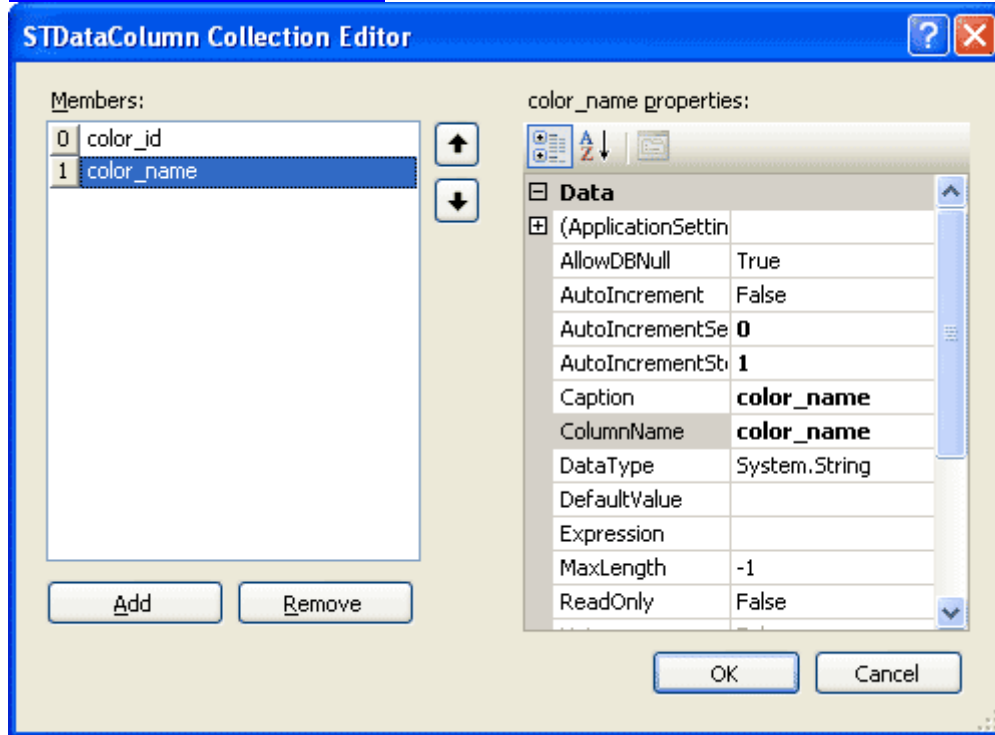
We have two tables: `COLORS` and `FLOWERS`. Suppose we want to edit `FLOWERS` table using lookup fields by selecting some color from Combobox that contains the list of colors from `COLORS` table. Isn't clear? No problems. It's more simply to show.

### ***Creating application***

In our application we will use table `COLORS` and `FLOWERS`. See the chapter [Using Complex Query in Adonet Oracle Data Components](#) to get information how to make them. Create a Windows Application using Visual Studio Wizard, just like in chapter [First Application](#).

Then drop to form STAdonetOracleConnection and rename it as stConnection and two STAdonetOraDataTable components and rename them as stAdonetOraDataTable1 and stColors correspondingly. We will use stColors as dictionary for lookup Combobox.

First add two columns to stColors component: COLOR\_ID (DataType: System.Int32) and COLOR\_NAME (DataType: System.String) using STDataColumn Collection Editor. If you don't know how to add columns see the chapter [Customising columns of STAdonetOracleDataTable](#). The columns collection of stColors should look like:

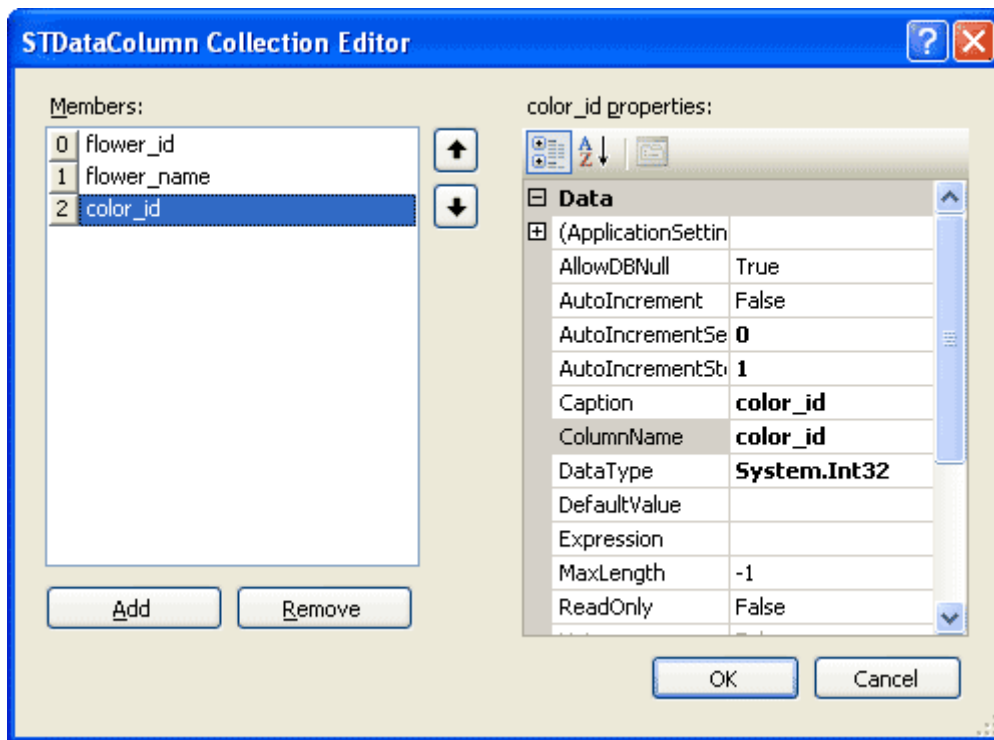


Set following properties of stColors:

**SQL** = select \* from colors;

**STAdonetOracleConnection** = stConnection;

Then go to stAdonetOraDataTable1. Add three columns: FLOWER\_ID (DataType: System.Int32), FLOWER\_NAME (DataType: System.String), COLOR\_ID (DataType: System.Int32). Columns of stAdonetOraDataTable1 should look like:



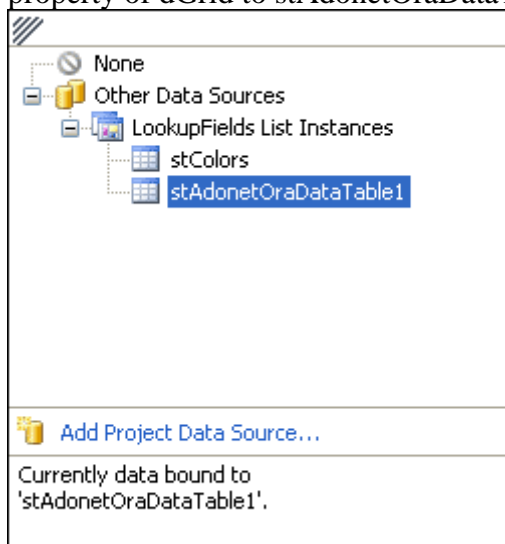
Set following properties of stAdonetOraDataTable1:

**RefreshOptions** = AfterInsert, BeforeEdit;

**SQL** = select t.\*, t.rowid from flowers t;

**STAdonetOracleConnection** = stConnection;

Then go to DataGridView component in our form and rename it to dGrid. Set **DataSource** property of dGrid to stAdonetOraDataTable1:



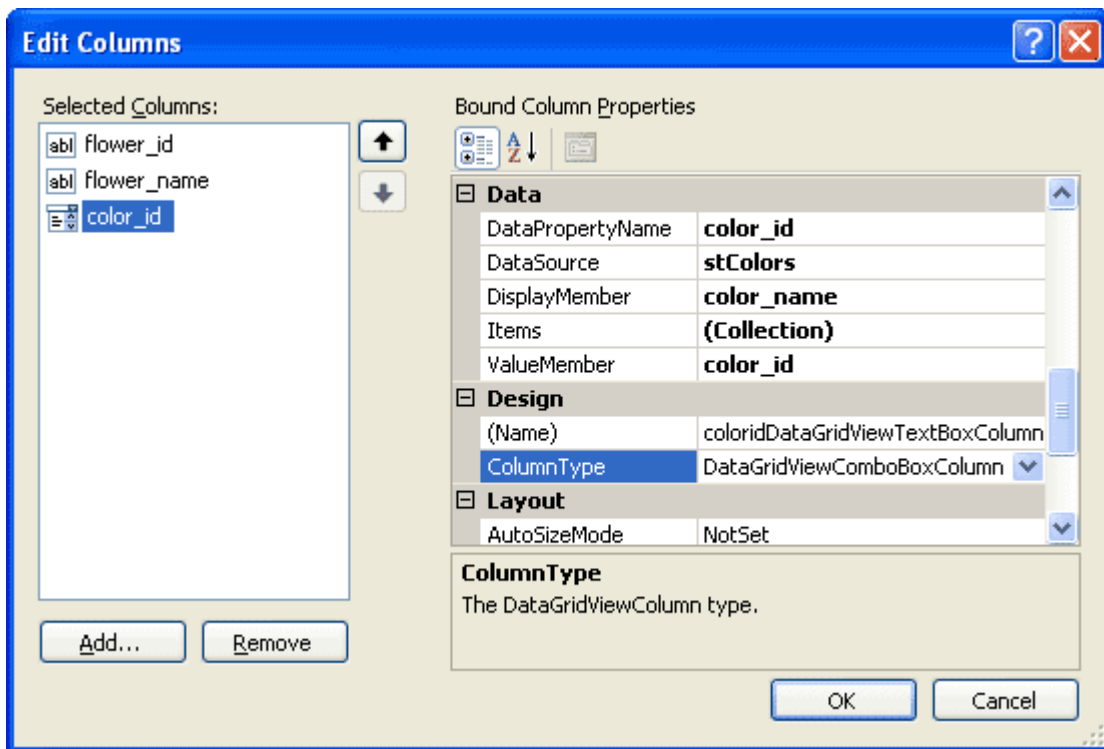
After that three columns should appear on our form in Visual Studio Designer. Then edit **Columns** property of dGrid component.

Change **ColumnType** property of color\_id to DataGridViewComboBoxColumn. Also set **DataPropertyName** = color\_id;

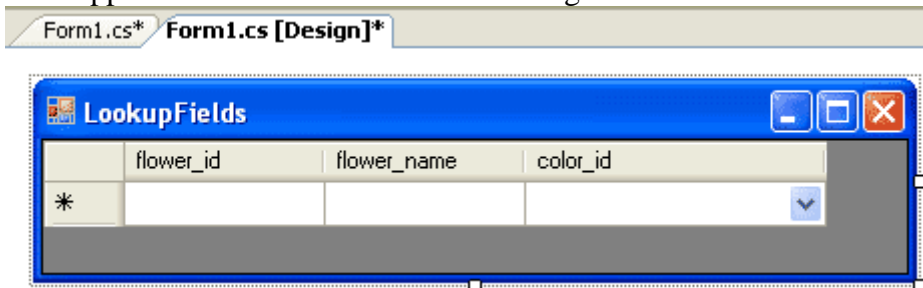
**DataSource** = stColors;

**DisplayMember** = color\_name;

**ValueMember** = color\_id;



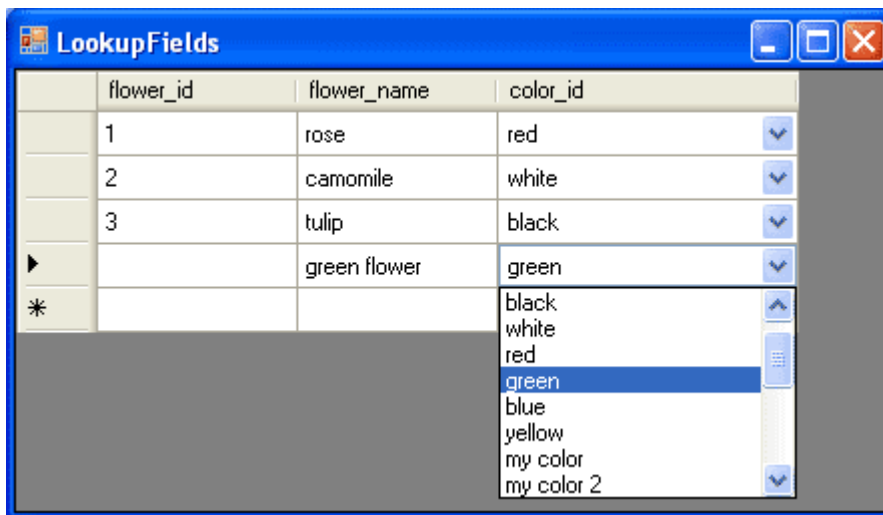
Now application form in Visual Studio designer should look like:



The code of our application will look like:

```
public LookupFields()
{
    InitializeComponent();
    try
    {
        stConnection.Open();
    }
    catch (Exception e)
    {
        MessageBox.Show(e.Message, "Error");
    }
    stColors.Open();
    stAdonetOraDataTable1.BindingControl = dGrid;
    stAdonetOraDataTable1.Open();
}
```

Compile and run application. Create your own flower, select color from ComboBox and post changes.



## Using Snotra Tech Adonet Oracle Data Components for console applications

Snotra Tech Adonet Oracle Data Components can be used not only for visual applications. They are suitable for any .NET applications and as an example for console programs.

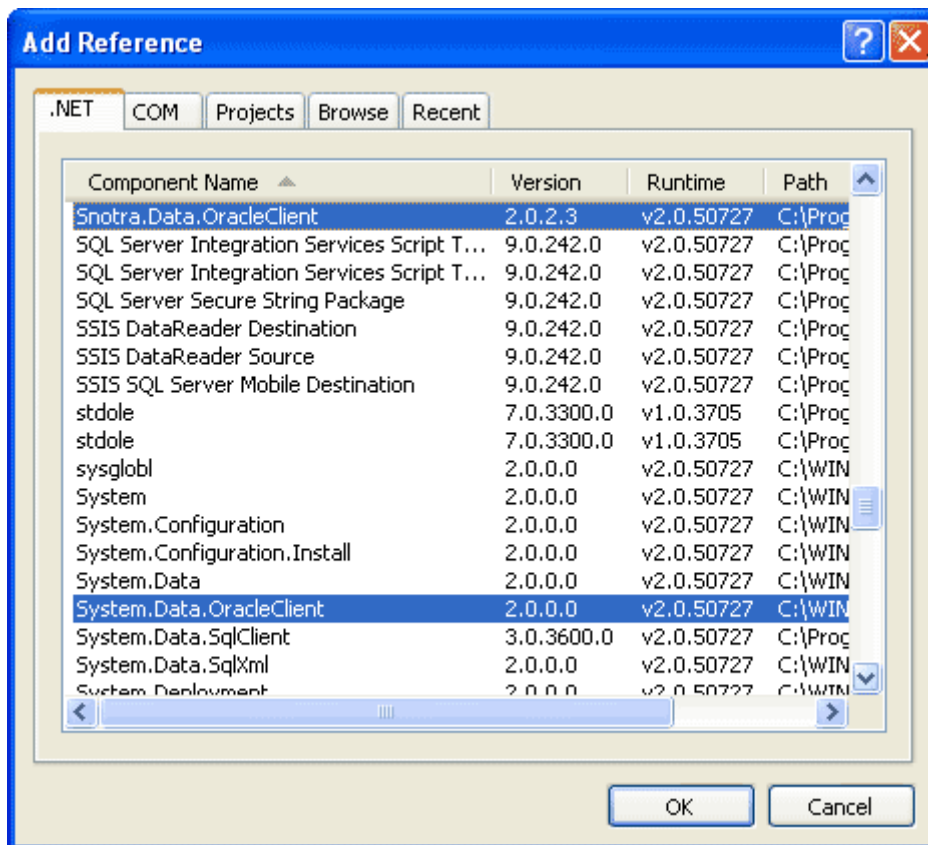
STAdonetOraDataTable has forward and backward iterators, so you can iterate rows easy. On can add new records manually, post changes in database, update delete, refresh one record, refresh all records and so on.

### ***Problem to solve***

Suppose we want to get all records from COLORS table. Then iterate records in STAdonetOraDataTable forward and backward. Then add a new records, post changes. Then refresh all records and iterate records forward.

### ***Creating application***

Create a new console application using Visual Studio 2005 wizard. After that one need to add two references:



Add following “using” directives in the beginning of the program:

```
using Snotra.Data.OracleClient;
using System.Data.OracleClient;
using Snotra.Data;
```

We will use OracleConnection class instead of STAdonetOracleConnection. They are interchangeable. Complete text of the program you can see below:

```
using System;
using Snotra.Data.OracleClient;
using System.Data.OracleClient;
using Snotra.Data;

namespace SimpleConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            OracleConnection conn = null;
            STAdonetOraDataTable datatable = null;
            try
            {
                conn = new OracleConnection("Data source=orcl;User Id=scott;Password=tiger");
                conn.Open();

                datatable = new STAdonetOraDataTable();
                datatable.ThrowExceptions = true;
                //allow add new records
                datatable.SQL = "select t.rowid, t.* from colors t";
                datatable.Connection = conn;

                datatable.Open();

                while (!datatable.Eof)
                {
                    STDataRow row = datatable[datatable.CurrentRecord];
                    Console.WriteLine(row["color_name"] + ", " + row["red"] + ", " + row["green"]
                        + ", " + row["blue"]);
                    datatable.Next();
                }
            }
        }
    }
}
```

```
    }
    Console.WriteLine("Then back...");
    while (!datatable.Bof)
    {
        STDataRow row = datatable[datatable.CurrentRecord];
        Console.WriteLine(row["color_name"] + ", " + row["red"] + ", " + row["green"]
            + ", " + row["blue"]);
        datatable.Prior();
    }
    //add a new row
    STDataRow r = datatable.AddNew();
    r["color_name"] = "MyPersonalColor";
    r["red"] = 25;
    r["green"] = 35;
    r["blue"] = 45;
    datatable.Post();
    Console.WriteLine("New record has been added successfully");
    //refresh datatable
    datatable.Refresh();
    //show all records
    while (!datatable.Eof)
    {
        STDataRow row = datatable[datatable.CurrentRecord];
        Console.WriteLine(row["color_name"] + ", " + row["red"] + ", " + row["green"]
            + ", " + row["blue"]);
        datatable.Next();
    }
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
finally
{
    if (datatable != null)
        datatable.Dispose();
    if (conn != null)
        conn.Dispose();
}
}
}
```

We use here Next() and Prior() functions to navigate records. Don't forget set `datatable.ThrowExceptions = true`. It allows to STAdonetOraDataTable throw exception if error is raised during some operation with Oracle DB.