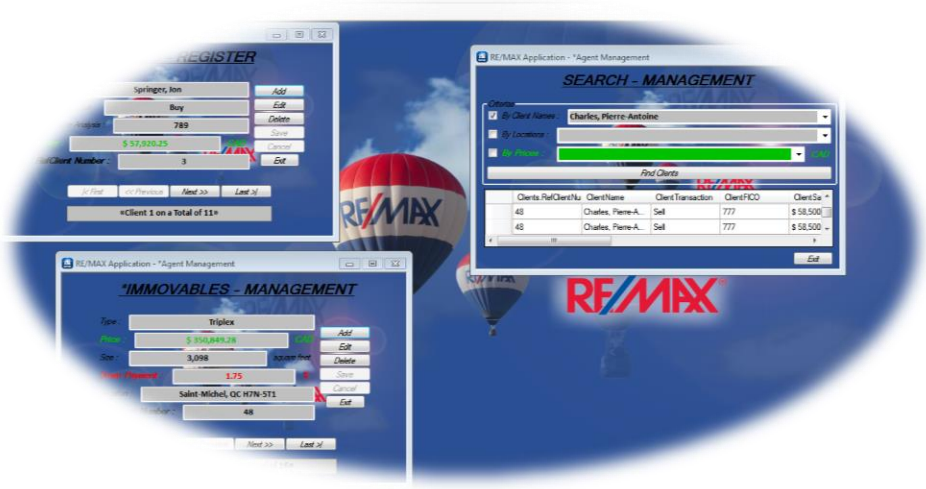
LaSalle College
Montréal




**Developing and Implementing the RE/MAX Canada Access 2013 – MS Application**

The project was developed in the Event Programming Java, JavaScript Course

at LaSalle College Montreal

By

Pierre Charles

The project is being presented to **IT Recruiting Specialists** at LinkedIn & LCI Education Network

to obtain an **Administrative Data Processing** (420.AO DEC) position

The project was supervised by the following Instructor:

Fodé Touré, On-Site Supervisor

# Table of Contents

# Acknowledgement

It was with much support that I have been able to complete this project and I would like to express my thanks to all involved. First and foremost, I would like to thank my On-Site Supervisor in the Event Programming Java, JavaScript Course, Instructor Touré. If it were not for his willingness to take me on as a Programmer of the **RE/MAX Canada Access 2013 – MS Application** (a three-tiered client/server environment), none of this would have been possible. Though I may have internally questioned his initial approach towards the project while providing me input on the layers designs and leaving me all alone to complete parts of it, I was at time overwhelmed. On the other hand, I'm extremely grateful for it now! It was during the sixth semester when I was stretched beyond my own comfort zone, and capabilities, when I understood what Programmers deal with on a daily basis, and when I recently figured how I could fit into this very complex IT Industry.

In addition, I would like to personally thank you the IT Recruiting Specialist for accepting to view this detailed technical report. Its this support that enabled me to do what I did in preparation for your presented role. As a result, I thought it would be beneficial for me to take advantage of this opportunity to disclose some of the knowledge I acquired during my pursuit for a Computer Science Diploma (420.AO DEC) in Business & Technologies at LaSalle College Montreal by presenting my involvement in the **RE/MAX Canada Access 2013 – MS Application** project.

# Introductions

**Project Motivation and Context**

The client was experiencing issues with data integrity while tracking their inventory of immovables within the Greater Montreal Area. The client required a new convenient process in order to create, read, update and delete records while maintaining the highest levels of data integrity. This new process would allow **RE/MAX** Canada's Real Estate Agents to keep accurate records of their clients (buyers and sellers), immovables and search registry base, in order to prevent re-occurrences with data corruption or unforeseen incidents within their day-to-day brokerage operation.

**Company Description**

**RE/MAX**, is short for "Real Estate Maximums," an American International real estate company that operates through a franchise system. The company has held the number one market share in the United States and Canada since 1999 for residential transaction. **RE/MAX** has more than 100,000 Real Estate Agents in over 6,800 offices and operates in over 100 countries. **RE/MAX** was also founded in January 1973 by Dave and Gail Liniger. The company was established with a maximum commission concept; in fact, their Real Estate Agents would keep nearly all of their commissions and only pay their brokerage share of the office expenses, rather than a share of the commission in order to maximize earnings.

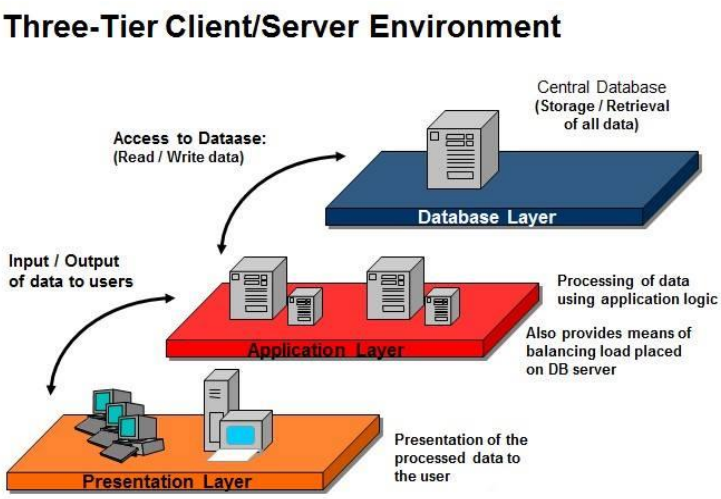
**Programmer, Course and Departmental Description**

The world of technologies is in a constant state of evolution. To meet the needs of the IT Industry, the Department of Computer Sciences at LaSalle College Montréal offered this Event Programming Java, JavaScript Course to train Entrepreneurs and IT Specialists like myself. Based on practical knowledge and problem-solving skills, the course provides the dynamics along with all the preparation required to enter our IT Industry. Through activities and close collaboration with several Industries, as an IT Specialist, I

receive practical training, allowing me to gain direct access to Enterprises. The Computer Science Technology Program (DEC), with the Administrative Data Processing Specialization stream, primed me to locate, collect and analyze data for the information processing requirements. I've learned how to achieve data modeling and data processing with all the appropriate development software. I'm also able to write operational and efficient programs utilizing a variety of programming languages, and oversee network planning and implementation, allowing for the secure sharing of end-user information.

**New System Description**

The **RE/MAX Canada Access 2013 – MS Application** is a complete client based application that help its end-users through the management of immovables within the Greater Montreal Area. The application program interface (API) utilizes the four basic functions of persistent storage known as C.R.U.D. (create, read, update and delete) to implement the relational database. This three-tiered client/server environment provides its end-users access to a tremendous amount of accurate data based on the buyer's and/or seller's needs. For instance, Real Estate Agents will now be properly equipped to analyze all data related to their buyer's and/or seller's specifications while preparing for weekly meetings. As a result, all immovable transactions will receive a greater sense of focus specific to their buyer's and/or seller's markets, in doing so Real Estate Agents improve their productivity in order to remain competitive.

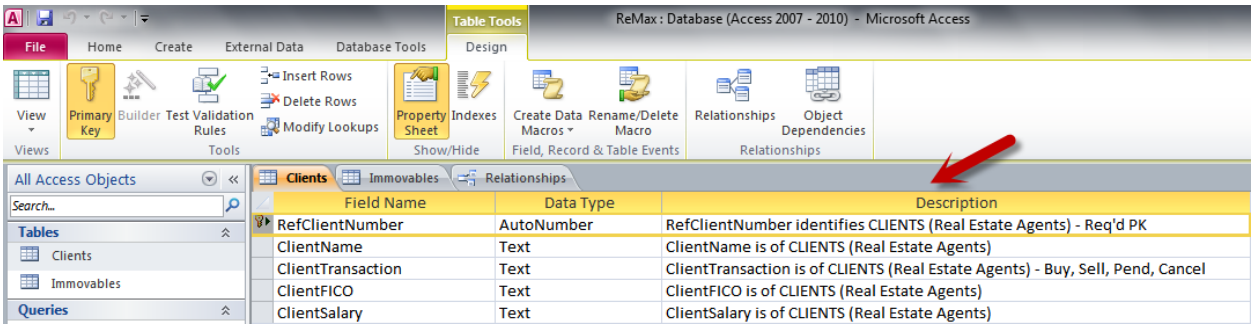



**Tools and Techniques Used**

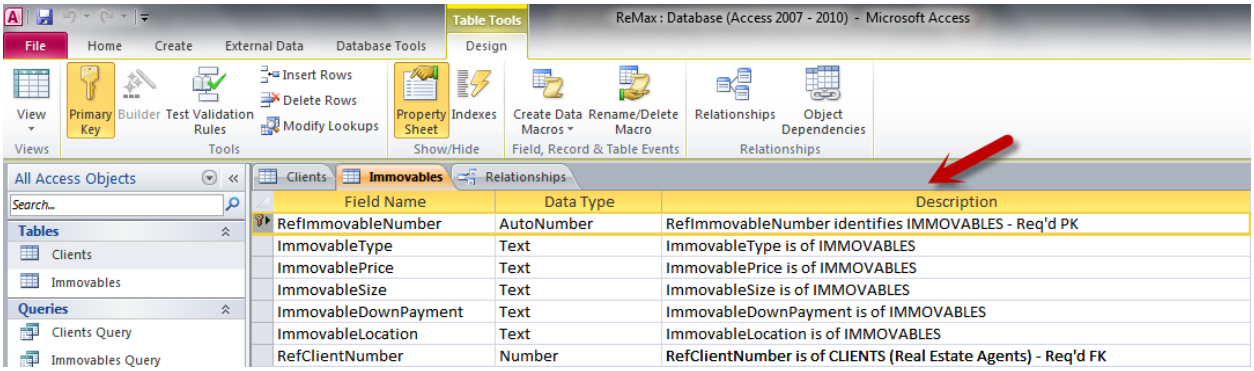
This subsection presents a small-scale description of the array of tools and techniques I used in the development and implementation of this project, such as but no limited to:

1. **Microsoft Access** 2010 is a database management system (DBMS) from Microsoft that combines the relational Microsoft Jet Database Engine with a graphical user interface (GUI) and software development tools that is also part of the Microsoft Office 2010 Suite of Applications. It can also import or link directly to data stored in other applications and databases. As a programmer, I used Microsoft Access to develop the database layer (storage/retrieval of all data). Like many other Office Applications, Access is supported by an object-based programming language that can reference a variety of objects including ActiveX Data Objects (ADO), and many other ActiveX components.

**Example 1.0:** Database Object – Table CLIENTS (Parent Table)

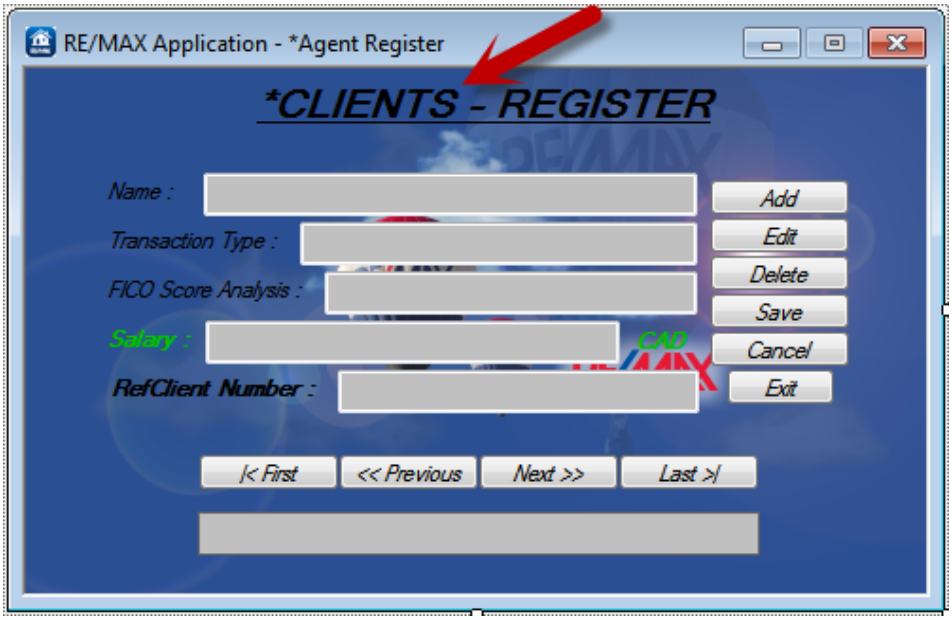



**Example 2.0:** Database Object – Table IMMOVABLES (Child Table)

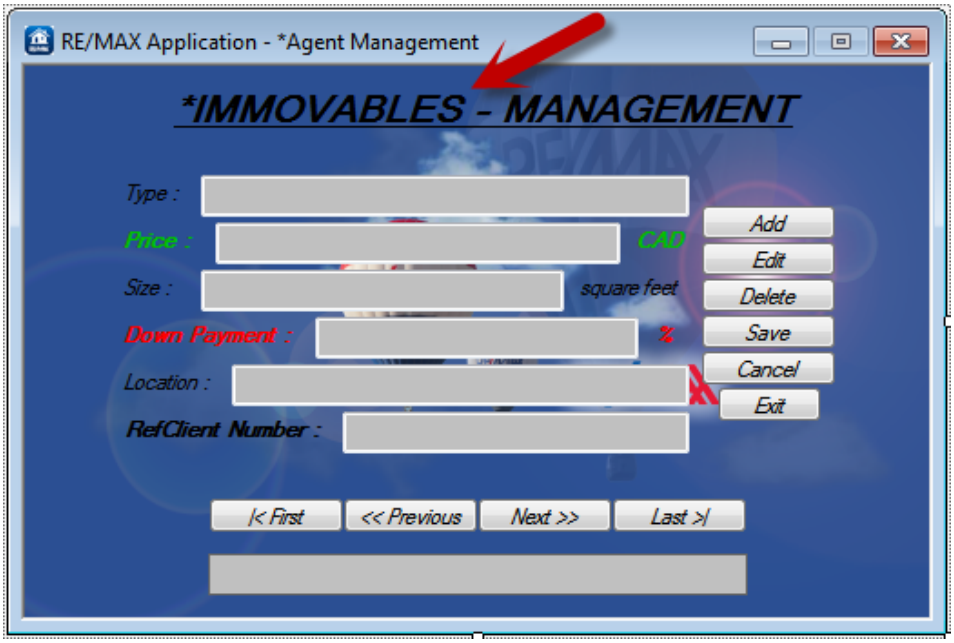


2. **Microsoft Visual Studio** 2012 is an integrated development environment (IDE) from Microsoft. Visual Studio includes a code editor supporting IntelliSense (the code completion component) as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a forms designer for building graphical user interface (GUI) applications, class designer, and database schema designer. As a programmer, I used it develop and implement this presentation layer (input/output of data to end-user) and business layer (access to read/write to database) for the **RE/MAX Canada Access 2013 – MS Application**.

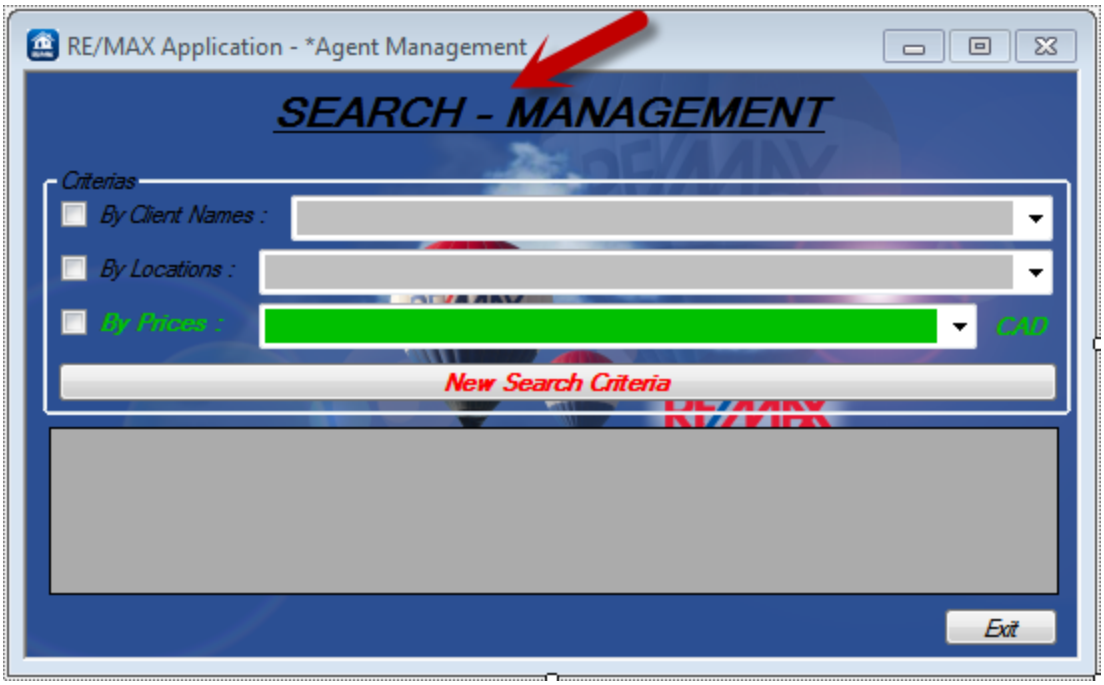
**Example 1.0:** frmClients.cs [class design] *CLIENTS – REGISTER




**Example 2.0:** frmImmovables.cs [class design] *IMMOVABLES – MANAGEMENT




**Example 3.0:** frmSearch.cs [class design] SEARCH – MANAGEMENT



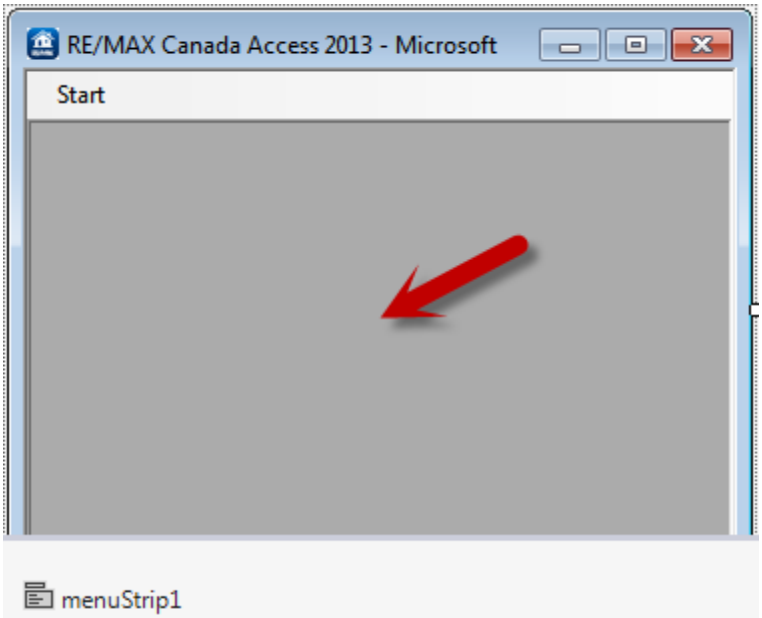

**The Main Tasks**

The first task of this development is implementing a relational database management system (DBMS)

suitable to execute structured query language (SQL) in order to navigate through the data stored while the end-user queries transactional data within the **RE/MAX Canada Access 2013 – MS Application**. This database design methodology will have its greatest impact on system elements like reliability, performance and security. And provided that, the system attributes are well developed and implemented within the entity relationship diagram (ERD) and relational data model (RDM) the system will be most effective. In addition, the database normalization implemented the following process, but was not limit to:

1. Creating and Refining current Business Requirements
2. Identifying and Refining the Classes of People, Places or Items
3. Defining and Refining current Relationships
4. Defining and Refining the Physical Data Types (*specifically for **Microsoft Access** 2010*)

The second task of this development is implementing the ActiveX Data Object (ADO) technology available within Microsoft Visual Studio 2012. As a programmer, I used the OleDbConnection Class of the System.Data.OleDb to request access to the Microsoft Access database in order to open a connection to the data source within the client based application's main form.



**Example 1.0:** frmMain.cs [Design] MAIN (shown on page 11 without property: BackgroundImage – System.Drawing.Bitmap)

## Problem Statement

**Current Design**

To reiterate, the **RE/MAX Canada Access 2013 – MS Application** is a complete client based application that will help its end-users through its complex management of their clients, immovables and search registry base. The API re-enforces the four functions of persistent storage to implement in the RDBMS. The database design combines all real estate intelligence within the Greater Montreal Area for the means of managing all transaction types within the day-to-day brokerage operations. In doing so, all its end-users will gain a substantial amount of information based on the client's requirements and demands.

This new client based application is set up to allow end-users like Quebec Real Estate Agents, to sign-in to their desktop and launch upon first arrival in order to access the Greater Montreal Area Real Estate Market. The end-user will then be able to modify (add, edit, delete, save, cancel and exit) any market transaction. Furthermore, the end-user will be able to search the specified criteria's by the client's name, location or price in order to allocate detailed data about an immovable like its type (condo, bungalow, duplex, triplex and etc.), size (square footage), down-payments (percentage) and reference client number (record) while analyzing one or several market transactions.

**Desired Functionalities**

In addition, this client base application is expected to manage the end-user's input into its relational database management system (RDBMS) through complex validation processes like the RefClientNumber found in the application layer (input/output of data to end-user) using all real estate and RDBMS

intelligence. As a result, all raw data found from table clients and table immovables will be much easier for Quebec Real Estate Agents to analyze when attending weekly meetings to discuss future marketing strategies, operation issues and market updates.

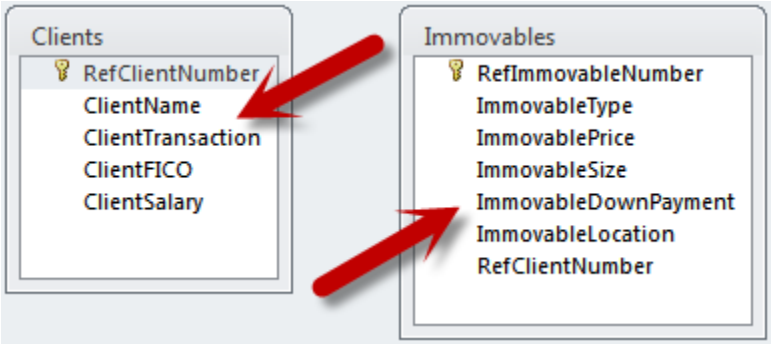
## Implementation

**Class Diagram**

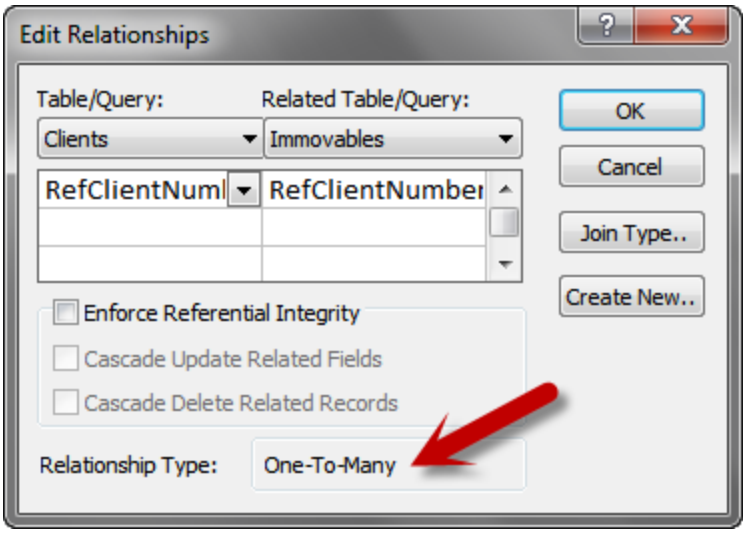
In order to create a functional client based application for both current and future demands, there will always be a need for research and development (R & D), from both a technological stand point and an end-user stand point. This Programming role consisted of capacity planning, configuration, database design, performance monitoring, security and troubleshooting, thus make it possible to lead into the best possible solution.

Step one, as a Programmer, we (other Programmers and On-Site Supervisor) developed a static structure diagram describing the system's classes, attributes, operations (or methods) and relationships among its objects with all the intelligence we gathered from **RE/MAX** Canada, like the people and items involved.

**Example 1.0:** Database Object – Table CLIENTS (*People*) and Table IMMOVABLES (*Items*)

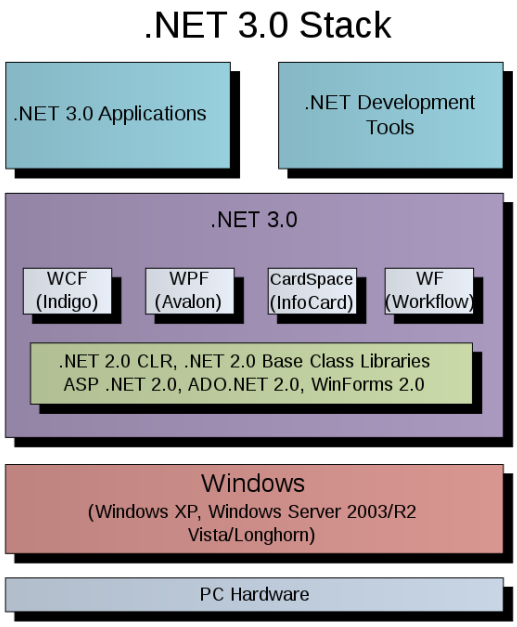



**Example 1.1:** Database Relationship – One-to-Many




**ADO.NET – ARCHITECTURE & O/R MAPPING**

To reiterate further, ADO.NET is a data access technology from the Microsoft .NET Framework that provides communication between this relational system through a common set of components. As a Programmer, ADO.NET provided a set of computer software components used to access data and data services from the database has part of the base class library. It is commonly used to access and modify data stored in relational systems and it is considered an evolution of ActiveX Data Object (ADO) technology. In addition, ADO.NET functionality exists within Microsoft Visual Studio IDE to create specialized subclasses of the DataSet classes for the database schema allowing convenient access to each field through strongly type properties. As a result, it helps catch programming errors at compile-time and enhances the IDE's Intellisense feature.

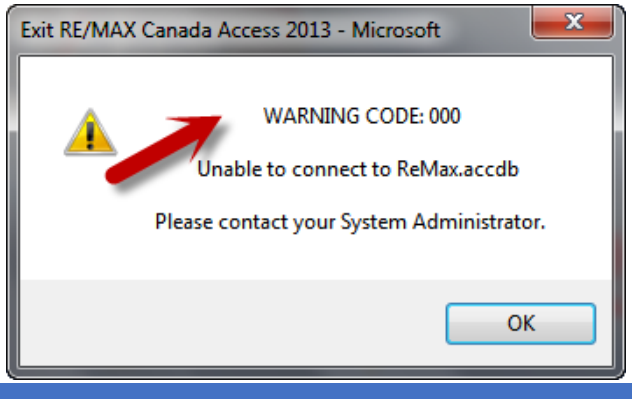


Step two, as previously discussed in the main task, as a Programmer, I developed and implemented the ActiveX Data Object (ADO) technology using the System.Data.OleDb Classes and the System.Data (the DataSet) in order to open a stable connection to the Microsoft Access database. In addition, I solely programmed a validation process in order to authenticate the application layer (read/write data to end-user) connection to its appropriate database and its supported window state property of MdiParent in order to manipulate the MdiChild windows of the presentation layer (input/output of data to end-user).

**Example 1.0:** Processing of data using Application Logic – Validation and Connection to Database

```
69          private void frmMain_Load(object sender, EventArgs e)
70          {
71              string database_accdb = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=E:\\_Desktop (2)\\P46_Lab (2)
72              string database_valid = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=E:\\_Desktop (2)\\P46_Lab (2)
73
74              // Database file name : E:\_Desktop (2)\P46_Lab (2)\prjCSWinReMax\prjCSWinReMax\Database\ReMax.accdb
75              clsGlobal.myCon = new OleDbConnection(database_accdb);
76              if (database_accdb != database_valid)
77              {
78                  // Then
79                  if (MessageBox.Show("\t      WARNING CODE: 000\n\n            Unable to connect to ReMax.accdb\r
80                  {
81                      Application.Exit();
82                  }
83              }
84              else
85              {
86                  clsGlobal.myCon.Open();
87
88                  OleDbCommand myCmd = new OleDbCommand("select * from CLIENTS", clsGlobal.myCon);
89                  clsGlobal.adpClients = new OleDbDataAdapter(myCmd);
90
91                  clsGlobal.mySet = new DataSet();
```

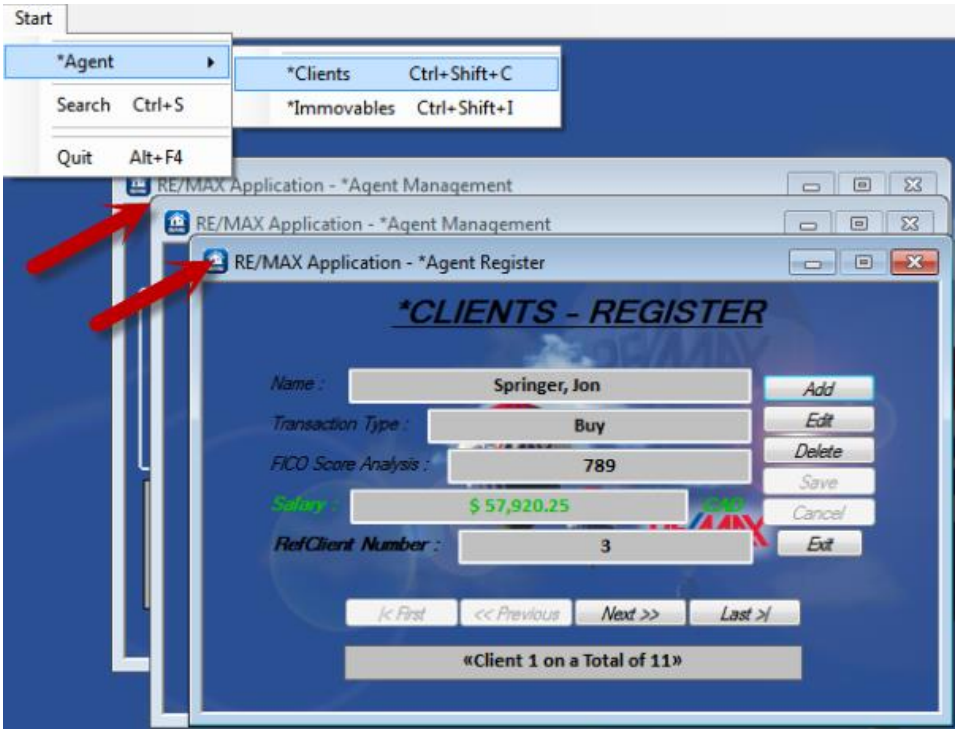
**Example 1.1:** Processing of data using Presentation Logic – Validation and Connection to Database – Warning message if unable to authenticate connect to Database: ReMax.accdb



**Example 2.0:** Processing of data using Application Logic – Creating MdiParent (agent registry window and agent management windows)

```
34          private void mnuClients_Click(object sender, EventArgs e)
35          {
36              /*frmClients fc = new frmClients();
37              fc.MdiParent = this;
38              fc.Show();*/
39
40              new frmClients() { MdiParent = this }.Show();
41          }
42
43          private void mnuImmovables_Click(object sender, EventArgs e)
44          {
45              /*frmImmovables fi = new frmImmovables();
46              fi.MdiParent = this;
47              fi.Show();*/
48
49              new frmImmovables() { MdiParent = this }.Show();
50          }
51
52          private void mnuSearch_Click(object sender, EventArgs e)
53          {
```

**Example 2.1:** Processing of data using Presentation Logic – Creating MdiParent (agent registry window and agent management windows)



Step three, picked-up from the recent creation of the MdiParent in the presentation layer, as Programmers we developed and implemented full read and write accesses. As a result of having implemented four separate forms called frmClients.cs, frmImmovables.cs, fromSearch.cs and their container called frmMain.cs, a very important global class called clsGlobal.cs is required in order to minimize the forms connection costs to the database layer (central database (storage/retrieval of all data)) for optimization. This very important global class handles all write and read accesses from all three forms into one static private variable connection process using OleDbConnection a vCon variable, two variable data adapters called OleDbDataAdapter a vClients and a vImmovables to handle the command builder during updates. Lastly, and most important of all, its static private DataSet variable called vSet (middleware methods) in order to finalize all updates from the database layer to the presentation layer in order to view all processed data to its end-user.

**Example 1:** Processing of data using Application Logic – Connection, Adapter and DataSet (Middleware source code in Application Layer (processed data used RAM prior to presentation in Presentation Layer))

```
22      class clsGlobal
23      {// Begin of class clsGlobal : handles read and write between all 3 forms (Clients, Immovables and Search).
24
25          static private OleDbConnection vCon;
26          //**DATA ADAPTERS -> Command Buidlers**
27          static private OleDbDataAdapter vClients;
28          static private OleDbDataAdapter vImmovables;
29          //**IMPORTANT**
30          static private DataSet vSet;
31          //=========================================myCon
32          static public OleDbConnection myCon
33          {
34              get { return clsGlobal.vCon; }
35              set { clsGlobal.vCon = value; }
36          }
37          //=========================================**DATA ADAPTERS -> Command Buidlers**
38          static public OleDbDataAdapter adpClients
39          {
40              get { return clsGlobal.vClients; }
41              set { clsGlobal.vClients = value; }
42          }
43
44          static public OleDbDataAdapter adpImmovables
```

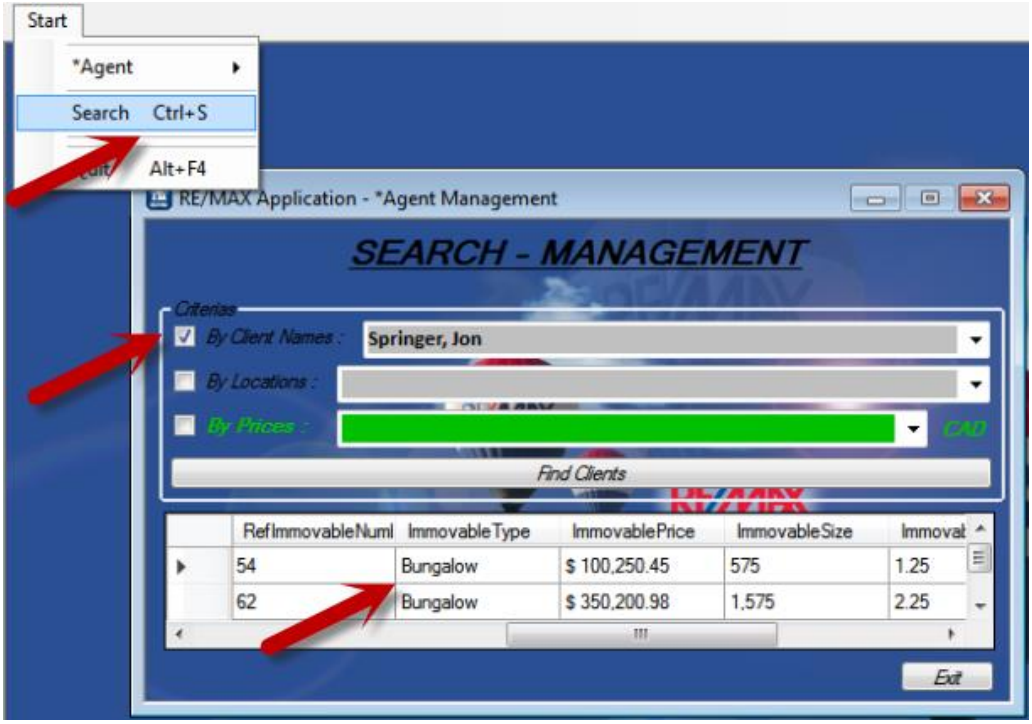
To reiterate, the end-user will then be able to modify any record using the add, edit, delete, save, cancel and exit methods, and read any record using the first, previous, next and last position methods from both the *CLIENTS – REGISTER and *IMMOVABLES – MANAGEMENT forms.

Step four, picked-up from the primary static private variable connection process using OleDbConnection a vCon variable which also used System.Data.OleDb to request global access to its Microsoft Access database in order to open a singular complex connection to the data source within the client based application's main form. Likewise, the search form called frmSearch.cs also used System.Data.OleDb in order to process a structure query language (SQL) command to search for the client names, locations and prices associated to all records (or immovables) in the client based application. Unlike the other two forms called frmClients.cs and frmImmovables.cs, the search form (or frmSearch.cs) uses the OleDbDataReader instead of OleDbDataAdapter because its SQL queries are set to only read records versus update records from the Microsoft Access database.

**Example 1.0:** Processing of data using Application Logic – Connection, Command and Reader

```
32          private void frmSearch_Load(object sender, EventArgs e)
33          {
34              btnNewSearch.Visible = false;
35
36              // Database Path : /prjCSWinReMax/prjCSWinReMax/Database/ReMax.accdb
37              clsGlobal.myCon = new OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0;Data Source=E:\\_Desktop (2
38              clsGlobal.myCon.Open();
39
40              OleDbCommand myCmd = new OleDbCommand("select RefClientNumber, ClientName from Clients", clsGlobal.m
41              OleDbDataReader rdClients = myCmd.ExecuteReader();
42              while(rdClients.Read() == true)
43              {
44                  cmbBoxByClients.Items.Add(rdClients["ClientName"].ToString());
45                  cmbBoxByClients.Items.Add("↑ [ " + rdClients["RefClientNumber"].ToString() + " ]");
46              }
47              rdClients.Close();
48
49              myCmd = new OleDbCommand("select ImmovableLocation from Immovables", clsGlobal.myCon);// <- for MS A
50              rdClients = myCmd.ExecuteReader();
51              while (rdClients.Read() == true)
52              {
53                  cmbBoxByLocations.Items.Add(rdClients["ImmovableLocation"].ToString());
54              }
```

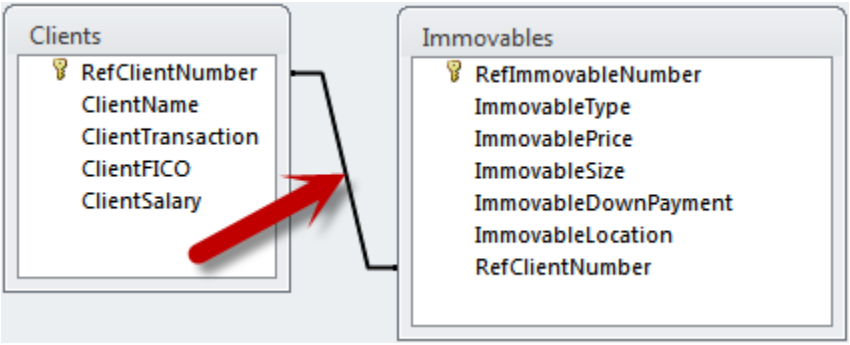
**Example 1.1:** Processing of data using Presentation Logic – Connection, Command and Reader




After one-of-many class meetings, as Programmers, we decided to program and implement the search form to allow its end-user to execute multiple queries at one time; on the contrary, the **RE/MAX Canada Access 2013 – MS Application** only managed the execution of one-out-of-three criteria's (or queries) during run time due to our implementation. As a Programmer, I began troubleshooting on my own time and realized our database layer was developed has a static structure diagram describing the system with only two tables with a one-to-many relationship. As a result, my analysis led me to believe we should have programmed and implemented the search form criteria's with three RadioButtons in order to allow its end-user to process only one search criteria at a time rather than up-to-three search criteria's at one time using the current CheckBoxes. The Application Logic (or source code) described all three CheckBoxes with a select all attributes from all tables, statement (SQL query 1: SELECT * FROM Clients, Immovables WHERE Clients.RefClientNumber = Immovables.RefClientNumber AND Clients.ClientName = ' " + cmbBoxByClients.SelectedItems.ToString() + " ';) using such SQL queries while implementing the

OleDbCommand to OleDbDataReader prior to data binding its result into the DataGridView does not rationalize the use of three CheckBoxes like my suggested three RadioButtons (see argument below).

**Example 2.0:** Database Object – Table CLIENTS (Parent Table) and Table IMMOVABLES (Child Table) and Database Relationship: One-to-Many – rationalization for implementing the suggested RadioButtons




**Example 3.0:** Processing of data using Application Logic – Check Box: Client Names, Locations and Prices – rationalization for implementing the suggested RadioButtons – source code: SQL query 1 and 2

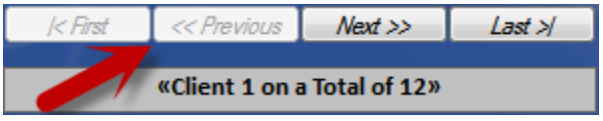
```
92              if(chkBoxByClients.Checked == true)
93              {
94              // mySQL = "select Clients.RefClientNumber, Clients.ClientName, Clients.ClientTransaction, Clients.Cl
95              mySQL = "select * from Clients, Immovables where Clients.RefClientNumber = Immovables.RefClientNumber
96              OleDbCommand myCmd = new OleDbCommand(mySQL, clsGlobal.myCon);
97              OleDbDataReader rdClients = myCmd.ExecuteReader();// <- for Access DATAREADER.
98              DataTable _TempTable = new DataTable();
99              _TempTable.Load(rdClients);
100             dataGridViewResult.DataSource = _TempTable;
101             }
102         }
103         else if (chkBoxByLocations.Checked == true)
104         {
105             //=======================================Checked Mode : Enabled & Disabled
106             chkBoxByClients.Enabled = false;
107             cmbBoxByClients.Enabled = false;
108             chkBoxByPrices.Enabled = false;
109             cmbBoxByPrices.Enabled = false;
110             lblImmovablePriceCAD.Enabled = false;
111             //=======================================Checked Mode : Visible & Invisiable
112             btnNewSearch.Visible = true;
113             //=======================================Process: Checked Locations
114             mySQL = "select * from Immovables, Clients where Immovables.RefClientNumber = Clients.RefClientNumber
```

**Example 3.1:** Processing of data using Application Logic – Check Box: Client Names, Locations and Prices – rationalization for implementing the suggested RadioButtons – source code: SQL query 3

```
121             else if (chkBoxByPrices.Checked == true)
122             {
123                 //=======================================Checked Mode : Enabled & Disabled
124                 chkBoxByLocations.Enabled = false;
125                 cmbBoxByLocations.Enabled = false;
126                 chkBoxByClients.Enabled = false;
127                 cmbBoxByClients.Enabled = false;
128                 //=======================================Checked Mode : Visible & Invisiable
129                 btnNewSearch.Visible = true;
130                 //=======================================Process: Checked Prices
131                 mySQL = "select * from Immovables, Clients where Immovables.RefClientNumber = Clients.RefClientNumber
132                 OleDbCommand myCmd = new OleDbCommand(mySQL, clsGlobal.myCon);
133                 OleDbDataReader rdClients = myCmd.ExecuteReader();// <- for Access DATAREADER.
134                 DataTable _TempTable = new DataTable();
135                 _TempTable.Load(rdClients);
136                 dataGridViewResult.DataSource = _TempTable;
```

Step five, after one-of-many class meetings, we decided to continue by focusing on the end-user's interactions with every button in the forms called frmClients.cs, frmImmovables.cs and fromSearch.cs. This program and implementation required me, as a Programmer, to understand every event as a Quebec Real Estate Agent viewing or modifying records within the **RE/MAX Canada Access 2013 – MS Application**. For example, an end-user viewing records from the *CLIENTS – REGISTER or the *IMMOVABLES – MANAGEMENT form would only need access to specific enabled buttons upon the forms' launch (or run time) like the next and last buttons.

**Example 1.0:** Processing of buttons using Presentation Logic – *CLIENTS – REGISTER form – enabled next and last buttons




**Example 1.1:** Processing of buttons using Application Logic – *CLIENTS – REGISTER form – enabled next and last buttons

```
118         private void btnPrevious_Click(object sender, EventArgs e)
119         {
120             // Condition restricts position from exceeding 0 (btnFirst_Click)
121             if(position > 0)
122             {
123                 position -= 1;
124                 Display();
125             }
126             //=========================================Click() Mode : Enabled & Disabled
127             btnFirst.Enabled = true;
128             btnNext.Enabled = true;
129             if(position == dataTbClients.Rows.Count - 2)
130             {
131                 btnNext.Enabled = false;
132             }
133             btnPrevious.Enabled = true;
134             if(position == 0)
135             {
136                 btnFirst.Enabled = false;
137                 btnPrevious.Enabled = false;
138             }
139             btnLast.Enabled = true;
140             // Condition restricts position from exceeing row count (btnLast_Click)
```

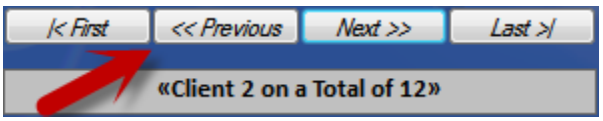
**Example 1.2:** Processing of buttons using Presentation Logic – *IMMOVABLES – MANAGEMENT form – enabled next and last buttons


**Example 1.3:** Processing of buttons using Application Logic – *IMMOVABLES – MANAGEMENT form – enabled next and last buttons

```
160         private void btnNext_Click(object sender, EventArgs e)
161         {
162             // Condition restricts position from exceeing row count (btnLast_Click)
163             if (position_ < (dataTbImmovables.Rows.Count - 1))
164             {
165                 position_ += 1;
166                 _Display();
167             }
168             //=========================================Click() Mode : Enabled & Disabled
169             btnFirst.Enabled = true;
170             btnNext.Enabled = true;
171             btnLast.Enabled = true;
172             if (position_ == dataTbImmovables.Rows.Count - 2)
173             {
174                 btnNext.Enabled = false;
175                 btnLast.Enabled = false;
176                 //=========================================Displaying : Click()
177                 txtPosition.Text = "«End of *Immovables ReMax.accdb database.»";
178             }
179             btnPrevious.Enabled = true;
180         }
181
182         private void btnLast_Click(object sender, EventArgs e)
```

Once the end-user begins navigating through the system's records the first and previous buttons become enabled allowing its end-user to navigate through all the system's records until the end-of-records. In fact, the end-user can choose to proceed viewing records by navigating backwards or forwards at this point.

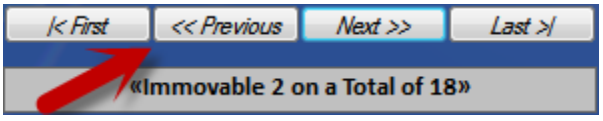
**Example 1.0:** Processing of buttons using Presentation Logic – *CLIENTS – REGISTER form – enabled first, previous, next and last buttons




**Example 1.1:** Processing of buttons using Application Logic – *CLIENTS – REGISTER form – enabled first, previous, next and last buttons

```
91          private void btnFirst_Click(object sender, EventArgs e)
92          {
93              position = 0;
94              Display();
95              //=========================================Click() Mode : Enabled & Disabled
96              btnFirst.Enabled = true;
97              btnNext.Enabled = true;
98              btnPrevious.Enabled = false;
99              btnLast.Enabled = true;
100             // Condition restricts position from exceeing row count (btnLast_Click)
101             if (position < (dataTbClients.Rows.Count - 1))
102             {
103                 btnAdd.Enabled = true;
104                 btnEdit.Enabled = true;
105                 btnDelete.Enabled = true;
106                 btnExit.Enabled = true;
107             }
108             if (position == 0)
109             {
110                 btnFirst.Enabled = false;
111             }
112             //=========================================Click() Mode : Visible & Invisible
113             lblClientReferenceNo.Visible = true;
```

**Example 1.2:** Processing of buttons using Presentation Logic – *IMMOVABLES – MANAGEMENT form – enabled first, previous, next and last buttons

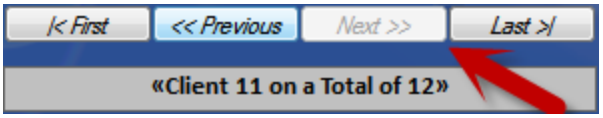


**Example 1.3:** Processing of buttons using Application Logic – *IMMOVABLES – MANAGEMENT form – enabled first, previous, next and last buttons

```
182         private void btnLast_Click(object sender, EventArgs e)
183         {
184             position_ = dataTbImmovables.Rows.Count - 1;
185             _Display();
186             //=========================================Displaying : Click()
187             txtPosition.Text = "«End of *Immovables ReMax.accdb database.»";
188
189             txtImmovableType.Text = "«Click Add to enter New *Immovable»";
190             txtImmovableRefClientNo.Text = "«Refer to *REGISTER»";
191             //=========================================Click() Mode : Enabled & Disabled
192             btnFirst.Enabled = true;
193             btnNext.Enabled = false;
194             btnPrevious.Enabled = true;
195             btnLast.Enabled = true;
196             btnEdit.Enabled = true;
197             if (position_ == dataTbImmovables.Rows.Count - 1)
198             {
199                 btnLast.Enabled = false;
200                 btnEdit.Enabled = false;
201             }
202             btnDelete.Enabled = false;
203             btnCancel.Enabled = true;
204         }
```

While the end-user is viewing the end-of-records, the forms' next button becomes disabled allowing its end-user to navigate through the remaining enabled buttons like previous or first or last in order to continue backwards navigation or view last record in order to add a new client or immovable the **RE/MAX Canada Access 2013 – MS Application**.

**Example 1.0:** Processing of buttons using Presentation Logic – *CLIENTS – REGISTER form – enabled first, previous and last buttons




**Example 1.1:** Processing of buttons using Application Logic – *CLIENTS – REGISTER form – enabled first, previous and last buttons

```
154         private void btnNext_Click(object sender, EventArgs e)
155         {
156             // Condition restricts position from exceeing row count (btnLast_Click)
157             if(position < (dataTbClients.Rows.Count - 1))
158             {
159                 position += 1;
160                 Display();
161             }
162             //=========================================Click() Mode : Enabled & Disabled
163             btnFirst.Enabled = true;
164             btnNext.Enabled = true;
165             if(position == dataTbClients.Rows.Count - 2)
166             {
167                 btnNext.Enabled = false;
168             }
169             btnPrevious.Enabled = true;
170             btnLast.Enabled = true;
171             // Condition restricts position from exceeing row count (btnLast_Click)
172             if (position < (dataTbClients.Rows.Count - 1))
173             {
174                 btnAdd.Enabled = true;
175                 btnEdit.Enabled = true;
176                 btnDelete.Enabled = true;
```

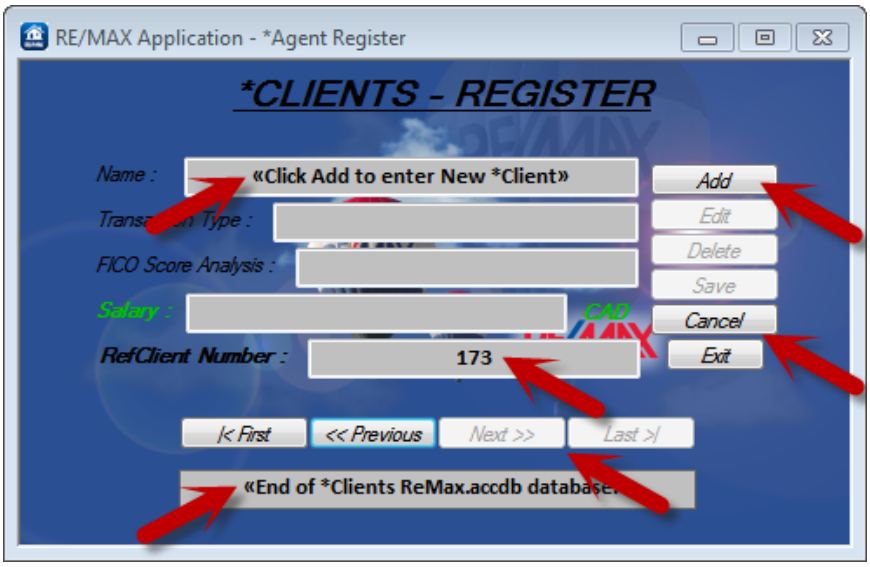
**Example 1.2:** Processing of buttons using Presentation Logic – *IMMOVABLES – MANAGEMENT form – enabled first, previous and last button


**Example 1.3:** Processing of buttons using Application Logic – *IMMOVABLES – MANAGEMENT form – enabled first, previous and last buttons

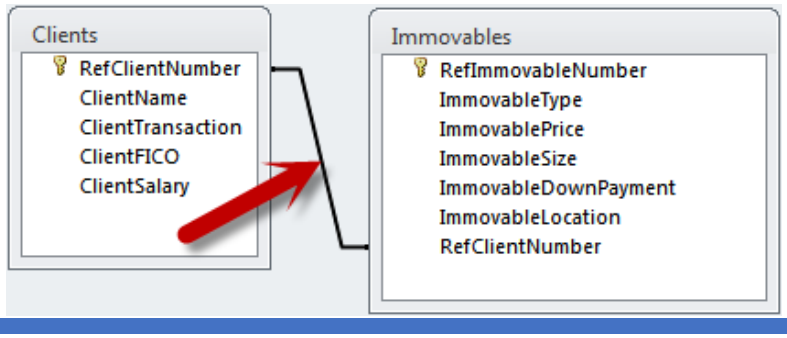
```
113         private void btnFirst_Click(object sender, EventArgs e)
114         {
115             position_ = 0;
116             _Display();
117             //=========================================Click() Mode : Enabled & Disabled
118             btnFirst.Enabled = true;
119             btnNext.Enabled = true;
120             btnPrevious.Enabled = false;
121             btnLast.Enabled = true;
122             if (position_ == 0)
123             {
124                 btnFirst.Enabled = false;
125             }
126         }
127
128         private void btnPrevious_Click(object sender, EventArgs e)
129         {
130             // Condition restricts position from exceeding 0 (btnFirst_Click)
131             if (position_ > 0)
132             {
133                 position_ -= 1;
134                 _Display();
135             }
```

Once the end-user decides to click on the last button, the next and last buttons become disabled while the display function notifies the end-user of end-of-records; and, to click on the add button via the display functions in order to add this new client or immovable into the system. A RefClient Number is also automatically generated in order to associate the database objects – from table client to table immovable. Meanwhile, the add, cancel and exit buttons have also enabled during this particular event.

**Example 1.0:** Processing of buttons using Presentation Logic – *CLIENTS – REGISTER form – enabled first and previous buttons for navigation and add, cancel and exit buttons for modifications

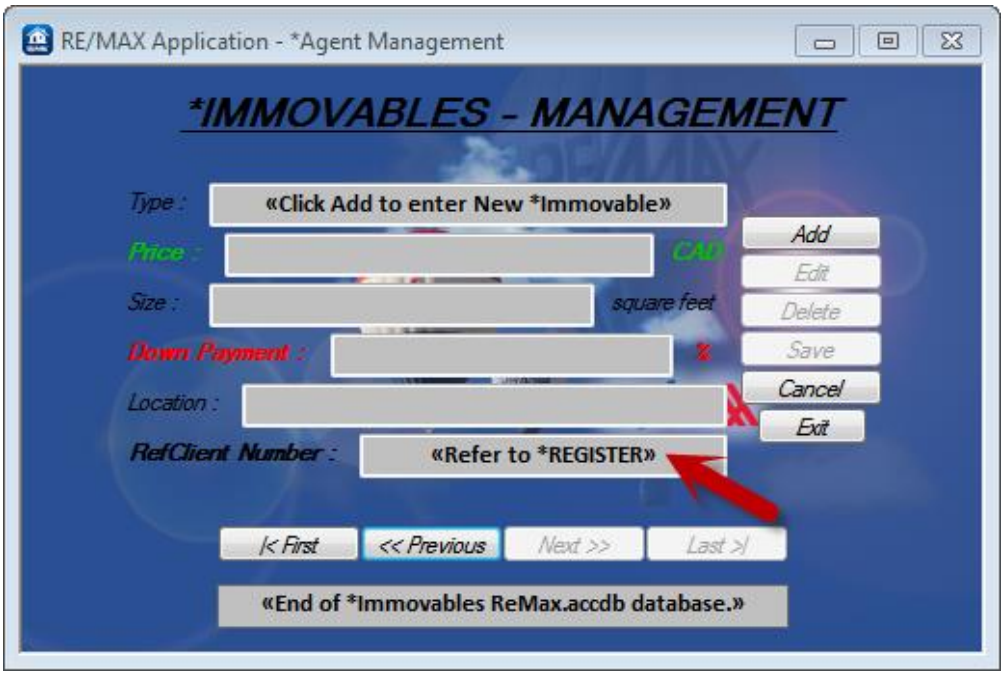



**Example 1.1:** Database Object – Table CLIENTS and Table IMMOVABLES and Database Relationship: One-to-Many



This event is considered complete once the end-user has successfully finished associating the new client to a new or existing immovable from the *IMMOVABLES – MANAGEMENT form (or Database Object – Table IMMOVABLES). Nonetheless, all the previously discussed buttons (or events) from the *CLIENTS – REGISTER form now share very similar attributes as the *IMMOVABLES – MANAGEMENT form.

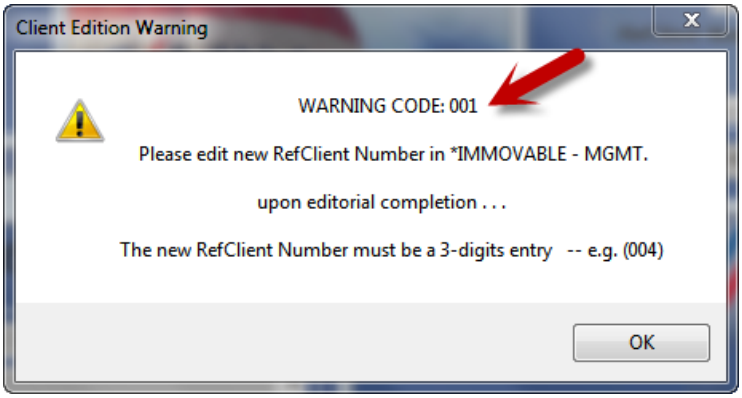
**Example 1.2:** Processing of buttons using Presentation Logic – *IMMOVABLES – MANAGEMENT form – the RefClient Number must be manually entered by the end-user during the add or edit events – enabled first and previous buttons for navigation and add, cancel and exit buttons for modifications




The end-user can now decide to click on the edit button in order to edit an existing immovable or client, the save and cancel buttons become enabled in order to save or cancel the current edition. Hence, the edit button from the *CLIENTS – REGISTER notifies the end-user via a warning message in order to edit the new or existing RefClient Number found in the *IMMOVABLES – MANAGEMENT form upon editorial

completion. By doing so, this event maintains the system's highest levels of data integrity for all records on behalf of all Quebec Real Estate Agents within the Greater Montreal Area (or Market).

**Example 1.0:** Edition of data using Presentation Logic – *CLIENTS – REGISTER – MessageBox, MessageBoxButton and MessageBoxIcon



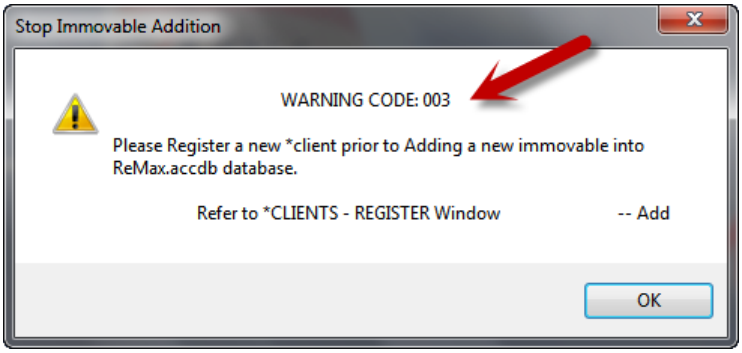

**Example 1.1:** Edition of data using Application Logic – *CLIENTS – REGISTER – MessageBox, MessageBoxButton and MessageBoxIcon

```
251             btnDelete.Enabled = false;
252             btnSave.Enabled = true;
253             btnCancel.Enabled = true;
254             btnExit.Enabled = false;
255             //=========================================Textbox State : Clear & Focus
256             txtClientName.Focus();
257             //=========================================Process: btnEdit()
258             MessageBox.Show("\t\tWARNING CODE: 001\n\n    Please edit new RefClient Number in *IMMOVABLE - MGMT
259             dataTbClients.Rows[position].BeginEdit();
260             // **Need an editing process for 'RefClientNumber'**
261             if(position >= 0)
262             {
263                 position -= 1;
264                 if(position < (dataTbClients.Rows.Count - 1))
265                 {
266                     position += 1;
267                 }
268             }
269             Display();
270             dataTbClients.Rows[position].Delete();
271             //=========================================Displaying : Edit()
272             txtPosition.Text = "«Editing an existing client...»";
273         }
```

The end-user can also decide to click on the add button in order to add a new immovable after a new client has been successfully registered into the *CLIENTS – REGISTER form, the *IMMOVABLES –

MANAGEMENT form will in turn notify its end-user of the addition event via both a display function and warning message in order to insure the new client registration.

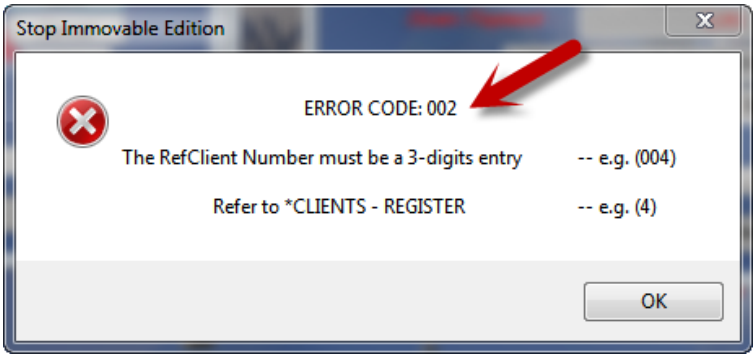
**Example 1.0:** Adding of data using Presentation Logic – *IMMOVABLES – MANAGEMENT – MessageBox, MessageBoxButton and MessageBoxIcon




**Example 1.1:** Adding of data using Application Logic – *IMMOVABLES – MANAGEMENT – MessageBox, MessageBoxButton and MessageBoxIcon

```
214
215             btnAdd.Enabled = true;
216             btnEdit.Enabled = false;
217             btnDelete.Enabled = false;
218             btnSave.Enabled = true;
219             btnCancel.Enabled = true;
220             btnExit.Enabled = false;
221             //=========================================Textbox State : Clear & Focus
222             txtImmovableType.Focus();
223             txtImmovableType.Clear();
224             txtImmovablePrice.Clear();
225             txtImmovableSize.Clear();
226             txtImmovableDownPayment.Clear();
227             txtImmovableLocation.Clear();
228             //=========================================Displaying : Add()
229             txtPosition.Text = "«Adding a new immovable...»";
230
231             txtImmovableRefClientNo.Text = "«Refer to *REGISTER»";
232
233             MessageBox.Show("\t\tWARNING CODE: 003\n\nPlease Register a new *client prior to Adding a new immo\
234             //=========================================Process: btnAdd()
235             dataTbImmovables.NewRow();
236         }
```

**Example 1.2:** Adding of data using Presentation Logic – *IMMOVABLES – MANAGEMENT – MessageBox, MessageBoxButton and MessageBoxIcon




**Example 1.3:** Adding of data using Application Logic – *IMMOVABLES – MANAGEMENT – MessageBox, MessageBoxButton and MessageBoxIcon – part 1 and 2

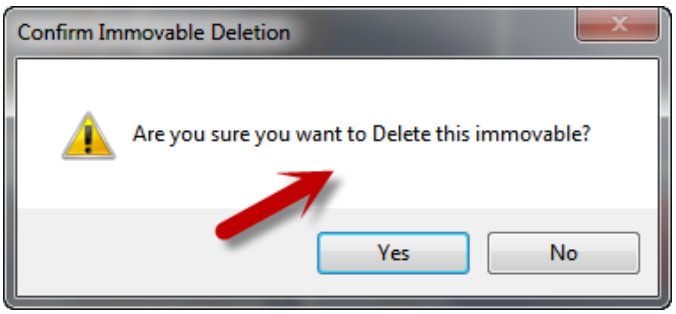
1.
```
34      private static class Validator
35      {
36          /// <summary>
37          /// This method validates if the input is a 3-digit number
38          /// </summary>
39          /// <param name="input"></param>
40          /// <returns>true if 3-digit number;otherwise, false</returns>
41          ///
42          public static bool IsValidNumber(string input)
43          {
44              if((Regex.IsMatch(input, @"^\d{3}$")))
45              {
46                  return true;
47              }
48              else
49              {
50                  return false;
51              }
52          }
53      }
```

2.
```
319          //=======================================Validation : RefClient Number
320          if(!Validator.IsValidNumber(txtImmovableRefClientNo.Text))
321          {
322              MessageBox.Show("\t\tERROR CODE: 002\n\nThe RefClient Number must be a 3-digits entry\t-- e.g.
323              txtImmovableRefClientNo.Text = " ";
324              txtImmovableRefClientNo.Focus();
325              return;
326          }
327          // **IMPORTANT** Binding NewRow() to DataRow.
328          DataRow _myRow;
```

The end-user can also decide to click on the delete button in order to delete an existing immovable or client from both the *IMMOVABLES – MANAGEMENT or the *CLIENTS – REGISTER forms, these forms will in turn notify their end-user of the deletion event via both a display function and warning message in order to confirm the immovable or client deletion. On the other hand, if their end-user should choose to reverse this deletion event, their end-user must click on the no button in order to begin its

cancellation process and then click the cancel button found in the *IMMOVABLES – MANAGEMENT or the *CLIENTS – REGISTER forms in order to return all pending data to their appropriate TextBoxes. As a result, the system will in turn confirm itself (the **RE/MAX Canada Access 2013 – MS Application**) and its end-user by notification of the cancelled update via the display functions.

**Example 1.0:** Deletion of data using Presentation Logic – *IMMOVABLES – MANAGEMENT – MessageBox, MessageBoxButton, MessageBoxIcon and DialogResult – steps 1, 2, 3 and 4

1. 


2. «Deleting an existing immovable...»

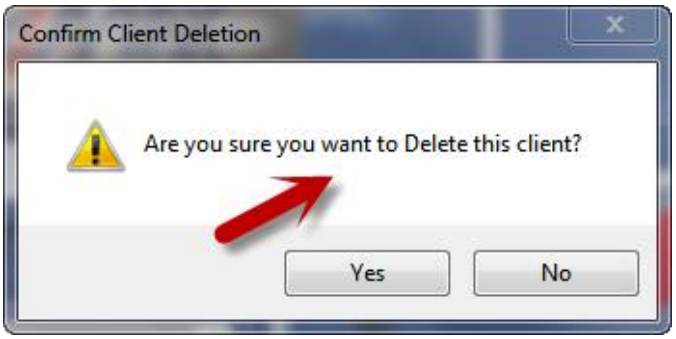
3. Cancel

4. «Cancelling update to ReMax.accdb immovable...»

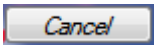
**Example 1.1:** Deletion of data using Application Logic – *IMMOVABLES – MANAGEMENT – MessageBox, MessageBoxButton, MessageBoxIcon and DialogResult

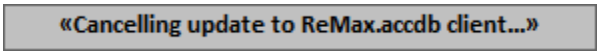
```
292             txtImmovableDownPayment.Clear();
293             txtImmovableLocation.Clear();
294             //=========================================Displaying : Delete()
295             txtPosition.Text = "«Deleting an existing immovable...»";
296             //=========================================Process: btnDelete()
297             if(MessageBox.Show("Are you sure you want to Delete this immovable?", "Confirm Immovable Deletion",
298             {
299                 if(position_ > 0)
300                 {
301                     position_ -= 1;
302                     if(position_ < (dataTbImmovables.Rows.Count - 1))
303                     {
304                         position_ += 1;
305                     }
306                 }
307                 dataTbImmovables.Rows[position_].Delete();
308 
309                 OleDbCommandBuilder myBuild = new OleDbCommandBuilder(clsGlobal.adpImmovables);
310                 clsGlobal.adpImmovables.Update(dataTbImmovables);
311 
312                 Delete_Display();
313             }
314         }
```

**Example 1.3:** Deletion of data using Presentation Logic – *CLIENTS – REGISTER – MessageBox, MessageBoxButton, MessageBoxIcon and DialogResult – steps 1, 2, 3 and 4

1. 


2. 


3. 

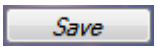

4. 


**Example 1.4:** Deletion of data using Application Logic – *CLIENTS – REGISTER – MessageBox, MessageBoxButton, MessageBoxIcon and DialogResult

```
294             txtFICOScoreAnalysis.Clear();
295             txtSalary.Clear();
296             //=========================================Displaying : Delete()
297             txtPosition.Text = "«Deleting an existing client...»";
298             //=========================================Process: btnDelete()
299             if(MessageBox.Show("Are you sure you want to Delete this client?", "Confirm Client Deletion", Messa
300             {
301                 dataTbClients.Rows[position].Delete();
302 
303                 OleDbCommandBuilder myBuild = new OleDbCommandBuilder(clsGlobal.adpClients);
304                 clsGlobal.adpClients.Update(dataTbClients);
305 
306                 if(position > 0)
307                 {
308                     position -= 1;
309                     if(position < (dataTbClients.Rows.Count - 1))
310                     {
311                         position += 1;
312                     }
313                 }
314                 Delete_Display();
315             }
316         }
```

The end-user can also decide to click on the save button in order to save a new or existing immovable or client from both the *IMMOVABLES – MANAGEMENT or *CLIENTS – REGISTER forms. As a result, the **RE/MAX Canada Access 2013 – MS Application** will also notify its end-user of the save event via the display functions.

**Example 1.0:** Saving of data using Presentation Logic – *IMMOVABLES – MANAGEMENT – System.Windows.Forms.Button – steps 1 and 2

1. 

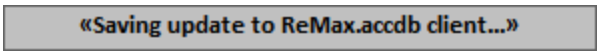

2. 


**Example 1.1:** Saving of data using Application Logic – *IMMOVABLES – MANAGEMENT – System.Windows.Forms.Button

```
349                 {
350                     position_ += 1;
351                 }
352             }
353             //=========================================Click() Mode : Enabled & Disabled
354             btnFirst.Enabled = true;
355             btnPrevious.Enabled = true;
356             btnNext.Enabled = true;
357             btnLast.Enabled = true;
358 
359             btnAdd.Enabled = true;
360             btnEdit.Enabled = true;
361             btnDelete.Enabled = true;
362             btnSave.Enabled = false;
363             btnCancel.Enabled = true;
364             btnExit.Enabled = true;
365             //=========================================Textbox States: Clear & Focus
366             txtImmovableType.Focus();
367             //=========================================Displaying : Save()
368             _Display();
369             //=========================================Display Save
370             txtPosition.Text = "«Saving an update to ReMax.accdb immovable...»";
371         }
```

**Example 1.2:** Saving of data using Presentation Logic – *CLIENTS – REGISTER – System.Windows.Forms.Button – steps 1 and 2

1. 


2. 

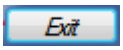

**Example 1.3:** Saving of data using Presentation Logic – *CLIENTS – REGISTER – System.Windows.Forms.Button

```
343                 {
344                     position += 1;
345                 }
346             }
347             //=========================================Click() Mode : Enabled & Disabled
348             btnFirst.Enabled = true;
349             btnPrevious.Enabled = true;
350             btnNext.Enabled = true;
351             btnLast.Enabled = true;
352 
353             btnAdd.Enabled = true;
354             btnEdit.Enabled = true;
355             btnDelete.Enabled = true;
356             btnSave.Enabled = false;
357             btnCancel.Enabled = true;
358             btnExit.Enabled = true;
359             //=========================================Textbox States: Clear & Focus
360             txtClientName.Focus();
361             //=========================================Displaying : Save()
362             Display();
363             //=========================================Display Save
364             txtPosition.Text = "«Saving update to ReMax.accdb client...»";
365         }
```

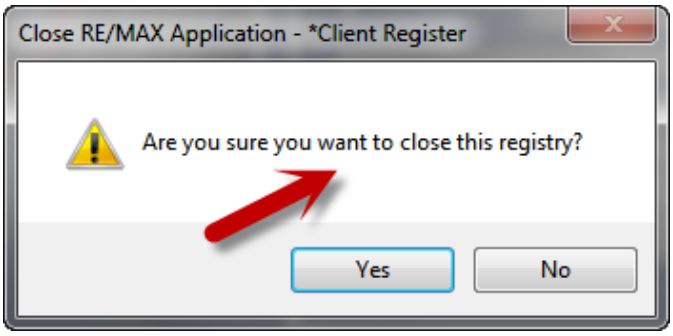
To reiterate, the end-user will now be able to cancel any of the previously discussed add, edit and delete events by simply clicking on the cancel button, in order to reverse one of the three events at a time. As a result, the **RE/MAX Canada Access 2013 – MS Application** will in turn also confirm itself by notifying its end-user of the cancelled update via the display functions.

The end-user can finally decide to click on the exit button in order to exit any of the three previously discussed forms called frmClients.cs, frmImmovables.cs and frmSearch.cs. As a result, the **RE/MAX Canada Access 2013 – MS Application** will also notify its end-user of the exit event via the display function and present a message box in order to confirm the end-user's form exit by clicking the yes or no button. On the contrary, the frmSearch.cs form implements the exact previously mentioned process except for its exit event notification process to its end-user or display function.

**Example 1.0:** Exiting of data using Presentation Logic – *CLIENTS – REGISTER – MessageBox, MessageBoxButton, MessageBoxIcon and DialogResult – steps 1, 2 and 3

1. 

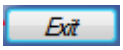

2. 


3. 

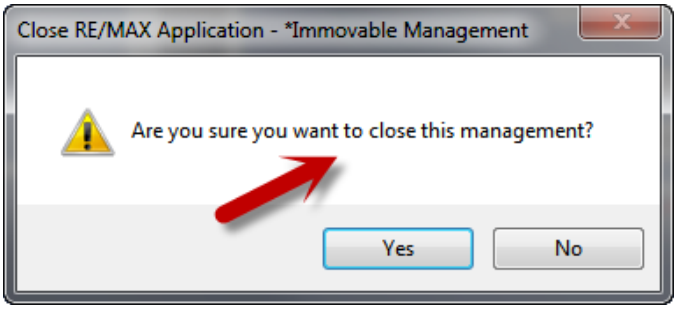

**Example 1.1:** Exiting of data using Application Logic – *CLIENTS – REGISTER – MessageBox, MessageBoxButton, MessageBoxIcon and DialogResult

```
397         private void btnExit_Click(object sender, EventArgs e)
398         {
399             //=======================================Display Exit
400             txtPosition.Text = "«Exiting ReMax.accdb client...»";
401
402             if (MessageBox.Show("Are you sure you want to close this registry?", "Close RE/MAX Application - *(
403             {
404                 this.Close();
405             }
406         }
407     }
408 }
409
```

**Example 1.2:** Exiting of data using Presentation Logic – *IMMOVABLES – MANAGEMENT – MessageBox, MessageBoxButton, MessageBoxIcon and DialogResult – steps 1, 2 and 3

1. 

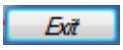

2. 

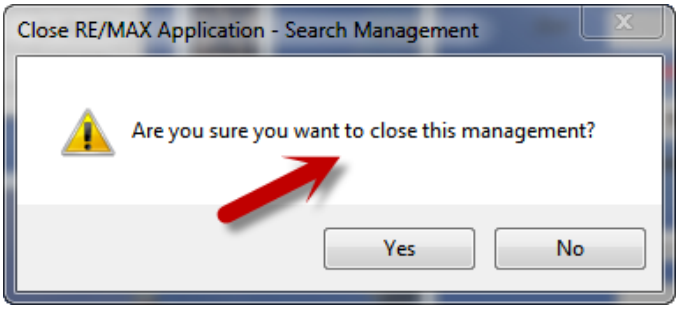

3. 


**Example 1.3:** Exiting of data using Application Logic – *IMMOVABLES – MANAGEMENT – MessageBox, MessageBoxButton, MessageBoxIcon and DialogResult

```
398         private void btnExit_Click(object sender, EventArgs e)
399         {
400             //=======================================Display Exit
401             txtPosition.Text = "«Exiting ReMax.accdb immovable...»";
402
403             if (MessageBox.Show("Are you sure you want to close this management?", "Close RE/MAX Application -
404             {
405                 this.Close();
406             }
407         }
408     }
409 }
410
```

**Example 1.4:** Exiting of data using Presentation Logic – SEARCH – MANAGEMENT – MessageBox, MessageBoxButton, MessageBoxIcon and DialogResult – steps 1 and 2

1. 


2. 

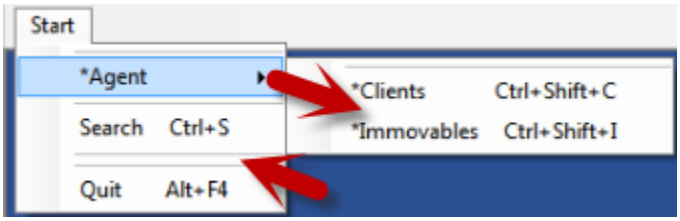

**Example 1.5:** Exiting of data using Application Logic – SEARCH – MANAGEMENT – MessageBox, MessageBoxButton, MessageBoxIcon and DialogResult

```
158         private void btnClose_Click(object sender, EventArgs e)
159         {
160             if (MessageBox.Show("Are you sure you want to close this management?", "Close RE/MAX Application -
161             {
162                 this.Close();
163             }
164         }
165     }
166 }
167
```

The end-user can also decide to click on the Quit Menu option or Alt+F4 short keys from the Start Menu option in order to quit the **RE/MAX Canada Access 2013 – MS Application** or the form called frmMain.cs. As a result, the **RE/MAX Canada Access 2013 – MS Application** will in turn notify its end-user of the quit event via a message box in order to confirm the end-user's form quit by clicking the yes or no button to either confirm the shut down execution or reverse the execution in order to power up.

Continuing with the Start Menu, the end-user can also decide to click on the Agent Menu option to access its Submenu in order to launch the *CLIENTS – REGISTER form by clicking on the *Clients Menu option or Ctrl+Shift+C short keys or launch the *IMMOVABLES – MANAGEMENT from by clicking on the *Immovables Menu option or Ctrl+Shift+I short keys. Lastly, the end-user can also decide to click on the Search Menu option or Ctrl+S short keys in order to launch the SEARCH – MANAGEMENT form.

**Example 1.0:** Start Menu options using Presentation Logic – *CLIENTS – REGISTER, *IMMOVABLES – MANAGEMENT, SEARCH – MANAGEMENT & Quit Menu option




**Example 1.1:** Start Menu options using Application Logic – *CLIENTS – REGISTER, *IMMOVABLES – MANAGEMENT, SEARCH – MANAGEMENT & Quit Menu option

```
53          private void mnuSearch_Click(object sender, EventArgs e)
54          {
55              /*frmSearch fs = new frmSearch();
56              fs.MdiParent = this;
57              fs.Show();*/
58
59              new frmSearch() { MdiParent = this }.Show();
60          }
61
62          private void mnuQuit_Click(object sender, EventArgs e)
63          {
64              if (MessageBox.Show("Are you sure you want to exit this applicaton?", "Exit RE/MAX Canada Access 2
65              {
66                  Application.Exit();
67              }
68          }
```

# Conclusion

In the preceding pages of my technical report, I as a Programmer, attempted to list the experience accumulated during this project by presenting my own personal and professional observations in preparation for this career building opportunity. This project was successfully built using different technologies and as been updated since June of 2015 as my own personal project. Using ADO.NET was challenging and moderate to implement as a C# programming language while establishing its communications between the client and database. In comparison to today's Enterprises (1000+ employees), Large (301-1000 employees), Mid-Size (101-300 employees) and Small (1-100 employees) Businesses, this project is inline with today's complexities has a three-tiered client/server environment. As a result, I believe my strong IT technical support expertise and recent college education make me an excellent candidate for this role and future IT roles. In fact, I have experience in learning new IT skill sets and continue excelling in new technologies. The key strengths I possess for success are excellent bilingual (English and French) communication and diagnostic skills, awesome background in applying technical and problem-solving skills while under pressure with an excellent history of ongoing research of products, procedures and best practices. Once again, I thank you for your time, and sincere consideration!

# Bibliography

Charles, P. (2015, June 2). *Google Drive (Early video version of Project)*, v10.0. Retrieved from LinkedIn - Business Network - Accomplishments: Win Re/Max Access Project (.mdb, .accdb, C#): https://drive.google.com/file/d/0B7p23IhLhUWvVlhaM0JBWHkyc2M/view