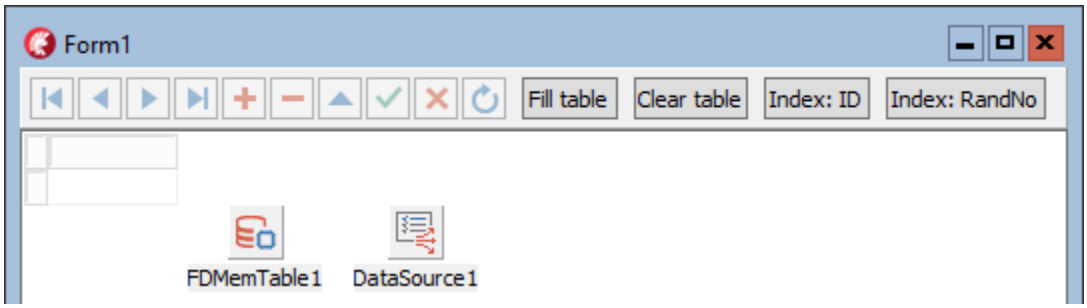
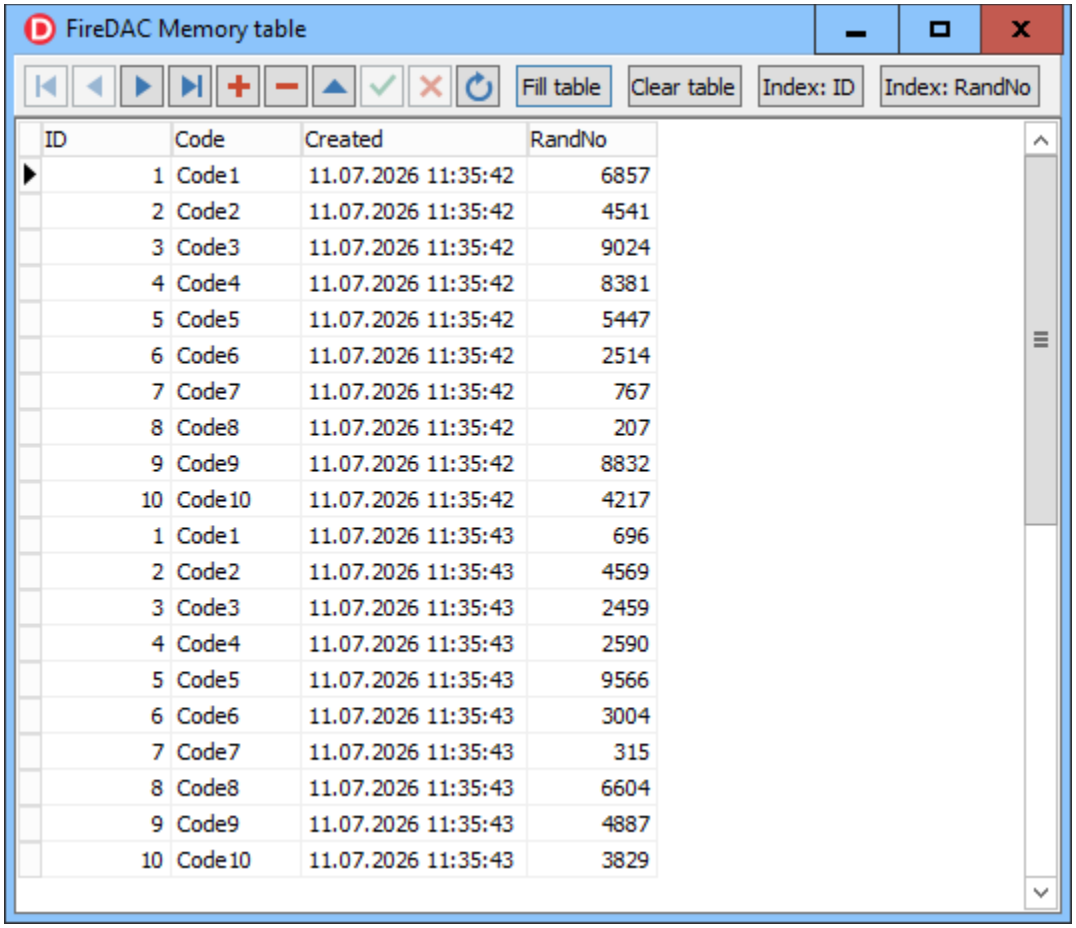
MemoryTables: FireDAC vs. ClientDataSet vs. LiteDAC

with Delphi 10.4.2

With Delphi FireDAC FDMemTable and ClientDataSet both may be used as in memory tables within an in memory database without any database connection and with a very similar application structure, and no need for specific database drivers.
The FireDAC memory table (FDMemTable) as well as the ClientDataSet this way accept what may be considered as generic (Delphi) field types.

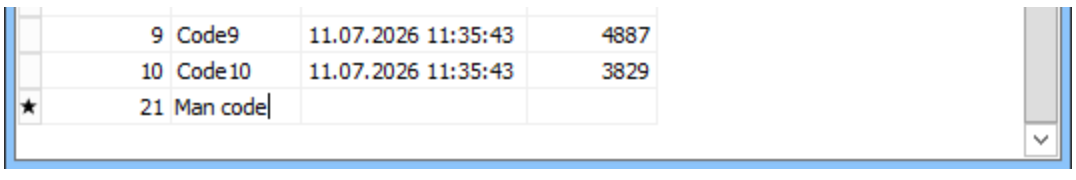
FDMemTable





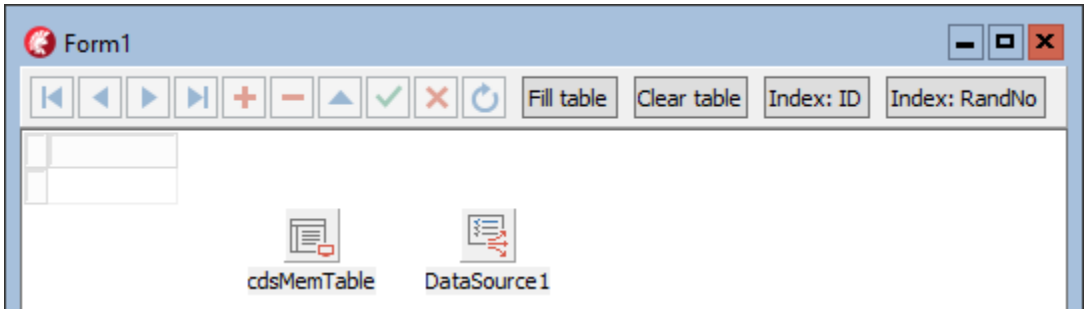
ID	Code	Created	RandNo
1	Code1	11.07.2026 11:35:42	6857
2	Code2	11.07.2026 11:35:42	4541
3	Code3	11.07.2026 11:35:42	9024
4	Code4	11.07.2026 11:35:42	8381
5	Code5	11.07.2026 11:35:42	5447
6	Code6	11.07.2026 11:35:42	2514
7	Code7	11.07.2026 11:35:42	767
8	Code8	11.07.2026 11:35:42	207
9	Code9	11.07.2026 11:35:42	8832
10	Code10	11.07.2026 11:35:42	4217
1	Code1	11.07.2026 11:35:43	696
2	Code2	11.07.2026 11:35:43	4569
3	Code3	11.07.2026 11:35:43	2459
4	Code4	11.07.2026 11:35:43	2590
5	Code5	11.07.2026 11:35:43	9566
6	Code6	11.07.2026 11:35:43	3004
7	Code7	11.07.2026 11:35:43	315
8	Code8	11.07.2026 11:35:43	6604
9	Code9	11.07.2026 11:35:43	4887
10	Code10	11.07.2026 11:35:43	3829

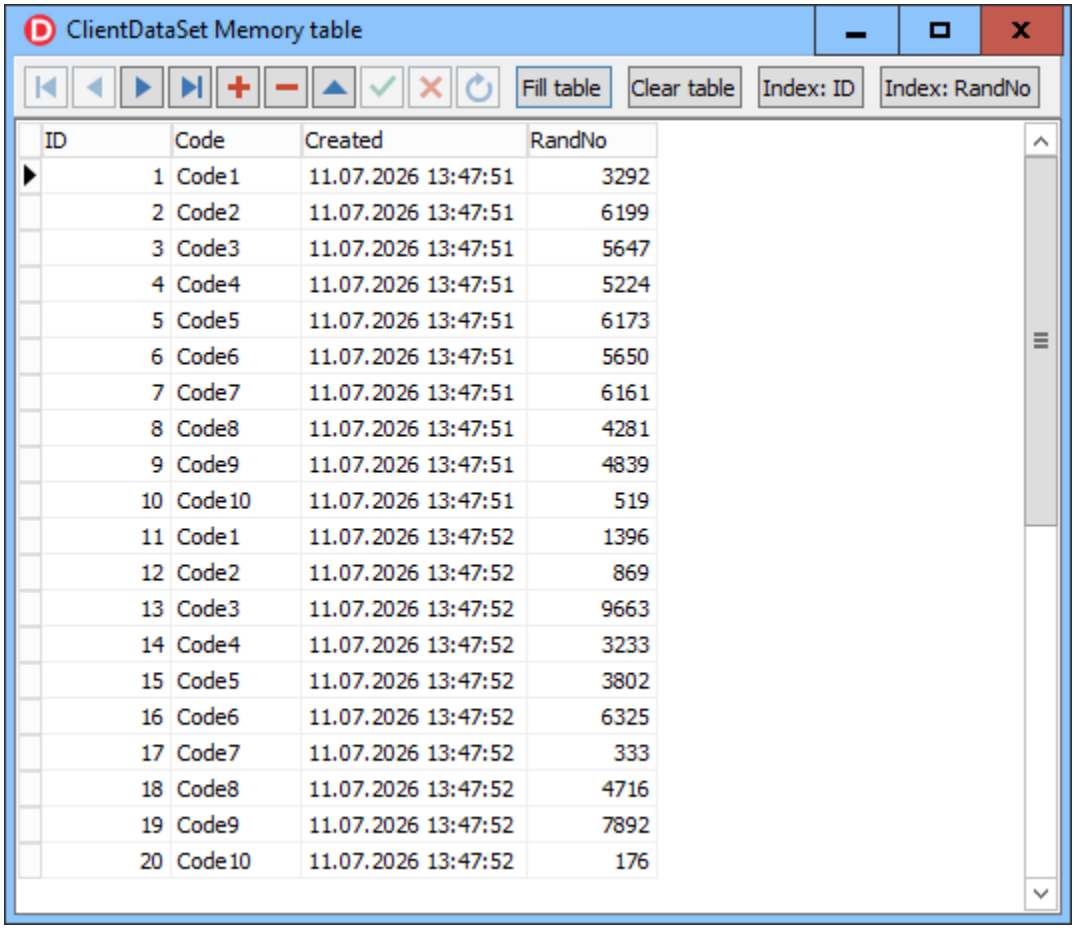
About the first field specified as Autoinc, appending records by code (ID field included) the Autoinc propagation is ignored, also accepting duplicate index field values (this is in contrast to ClientDataSet!) – however, manually adding records, the Autoinc ID field propagates properly:



9	Code9	11.07.2026 11:35:43	4887
10	Code10	11.07.2026 11:35:43	3829
★ 21	Man code		

ClientDataSet





ID	Code	Created	RandNo
1	Code1	11.07.2026 13:47:51	3292
2	Code2	11.07.2026 13:47:51	6199
3	Code3	11.07.2026 13:47:51	5647
4	Code4	11.07.2026 13:47:51	5224
5	Code5	11.07.2026 13:47:51	6173
6	Code6	11.07.2026 13:47:51	5650
7	Code7	11.07.2026 13:47:51	6161
8	Code8	11.07.2026 13:47:51	4281
9	Code9	11.07.2026 13:47:51	4839
10	Code10	11.07.2026 13:47:51	519
11	Code1	11.07.2026 13:47:52	1396
12	Code2	11.07.2026 13:47:52	869
13	Code3	11.07.2026 13:47:52	9663
14	Code4	11.07.2026 13:47:52	3233
15	Code5	11.07.2026 13:47:52	3802
16	Code6	11.07.2026 13:47:52	6325
17	Code7	11.07.2026 13:47:52	333
18	Code8	11.07.2026 13:47:52	4716
19	Code9	11.07.2026 13:47:52	7892
20	Code10	11.07.2026 13:47:52	176

FireDAC FDMemTable and ClientDataSet

Common for the FDMemTable and the ClientDataSet table is, that for common data field type as integer, float, string (VARCHAR, etc.), untyped binary, Boolean, Date/DateTime, the field types may be considered as generic Delphi fields.

Autoincrementing (key) fields

With ClientDataSet and FDMemTable (as well as Paradox and some other database formats) the Autoinc is available as a field type, and though for either you cannot specify this field as a Primary Key nor restricting values to be Unique, but for the ClientDataSet Autoinc field it implicitly behaves as if PRIMARY KEY and with Unique restriction without this being specified when created. For FDMemTable - see below.

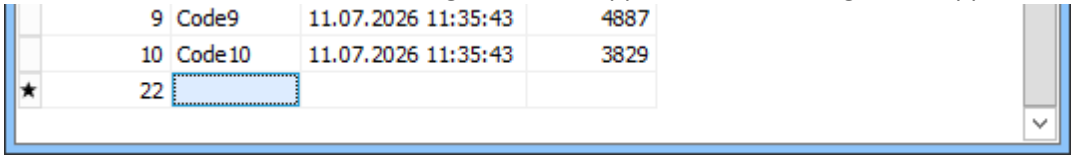
With many other databases the Autoincrement often rather has to be specified as a property of an integer field type (e.g. SQLite), or as a field associated trigger function (e.g. Interbase and FireBird).

Autoinc propagation (and FDMemTable issues)

Concerning the Autoinc, the ClientDataSet and the FDMemTable behaves differently, as the Autoinc of the ClientDataSet seems to be triggered by the OnPost (or BeforePost), whereas with FDMemTable the Autoinc propagation seems to be triggered on the Insert/Append event and this way independent of data inputs or the Post event.

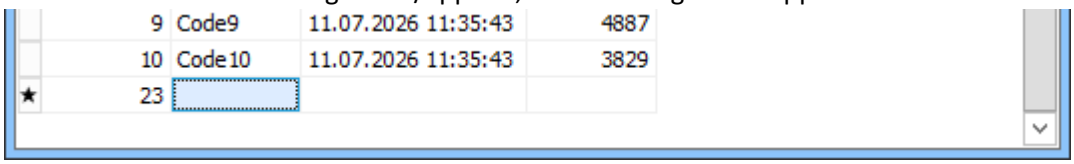
This way, the FDMemTable Autoinc counter values propagates when opening the Insert or Append record, but before entering any record data – also if cancelling the Insert or Append before Post, followed by a new Insert/Append, the Autoinc presents a new value disregarding the previous generated but non-saved Autoinc value. A very strange behavior!

FireDAC FDMemTable, after cancelling the record append, and initiating a new append:



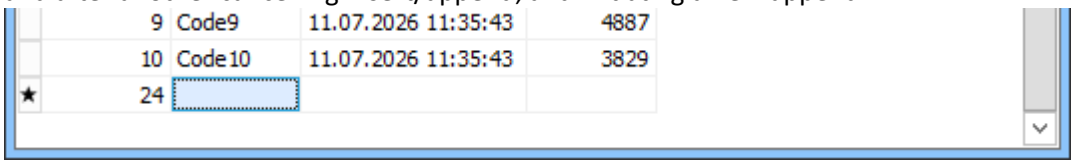
	9	Code9	11.07.2026 11:35:43	4887
	10	Code10	11.07.2026 11:35:43	3829
★	22			

and after another cancelling insert/append, and initiating a new append:



	9	Code9	11.07.2026 11:35:43	4887
	10	Code10	11.07.2026 11:35:43	3829
★	23			

and after another cancelling insert/append, and initiating a new append:



	9	Code9	11.07.2026 11:35:43	4887
	10	Code10	11.07.2026 11:35:43	3829
★	24			

SQL

Neither FDMemTable nor ClientDataSet can receive SQL queries.

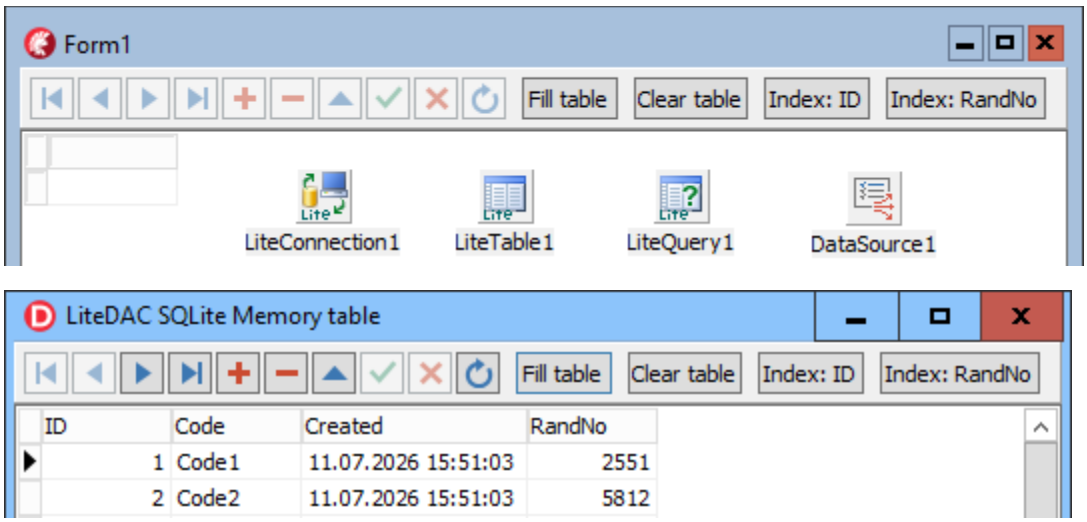
However FireDAC FDMemTable may receive SQL statements, if modified by adding an FDConnection (optionally as a in memory database), and adding FDLocalSQL as:

```
FDConnection2.DriverName := 'SQLite';      // has to be 'SQLite' ?
FDConnection2.DataBase := ':memory:';
FDLocalSQL1.Connection := FDConnection2;
FDLocalSQL1.DataSets.Add(FDMemTable1);
FDLocalSQL1.Active := True;
FDQuery1.SQL.Text := 'SELECT * FROM FDMemTable1 WHERE ... ORDER BY ...';
FCQuery1.Active := True;
```

Whatever the database, the SQL used by FDLocalSQL will be of the SQLite dialect.

LiteDAC – SQLite inMemory Database

Different from FDMemTable and ClientDataSet, the Devart LiteDAC does not provide a special memory table type but may be specified within an **in Memory database** (:MEMORY:) and works only with SQLite data and field types, where the Autoinc must be specified by an SQLIte dialect SQL creating an integer field with an associated PRIMARY KEY and Autoincrement field property to be specified.
Specified this way, the SQLite/LiteDAC Autoincremented field behaves just like the ClientDataSet Autoinc field, except raising error if trying to insert single records with specific values (e.g. via SQL INSERT INTO), though accepting bulk inserts via INSERT INTO TableName SELECT * FROM SourceTable.



This way, the LiteDAC memory database/table is to be specified as:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  strSql: string;
begin
  LiteConnection1.Database := ':MEMORY: ';
  LiteConnection1.Options.Direct := True;
  LiteConnection1.Connect;
  strSql := 'CREATE TABLE IF NOT EXISTS MemTable (';
  strSql := strSql + #13#10 + 'ID INTEGER PRIMARY KEY Autoincrement';
  strSql := strSql + #13#10 + ', Code VARCHAR(10)';
  strSql := strSql + #13#10 + ', Created DATETIME';
  strSql := strSql + #13#10 + ', RandNo INTEGER';
  strSql := strSql + #13#10 + ');';
  LiteQuery1.SQL.Text := strSQL;
  LiteQuery1.Execute;
  LiteTable1.TableName := 'MemTable';
  LiteTable1.Open;
  DataSource1.DataSet := LiteTable1;
  DBGrid1.DataSource := DataSource1;
  DBNavigator1.DataSource := DataSource1;
  Self.Caption := 'LiteDAC SQLite Memory table';
```

Also, with LiteDAC appending single records, you must (to avoid errors) omit inserting data into the Autoincremented PRIMARY KEY field, as this field type by LiteDAC is considered a read only field:

```
  for i := 1 to 10 do
    begin
      LiteTable1.Append;
//      LiteTable1.FieldByName('ID').AsInteger       := i;
      LiteTable1.FieldByName('Code').AsString      := 'Code'+IntToStr(i);
      LiteTable1.FieldByName('Created').AsDateTime := Now();
      LiteTable1.FieldByName('RandNo').AsInteger   := Random(10000);
      LiteTable1.Post;
    end;
```

/Niels Knabe
July 2026
www.nknabe.dk

```
unit uFDMemTable;

interface

uses
  Winapi.Windows
, Winapi.Messages
, System.SysUtils
, System.Variants
, System.Classes
, Vcl.Graphics
, Vcl.Controls
, Vcl.Forms
, Vcl.Dialogs
// adding TPanels:
, Vcl.ExtCtrls
// adding TButtons:
, Vcl.StdCtrls
// adding DBGrid & DataSource:
, Data.DB
, Vcl.Grids
, Vcl.DBGrids
// adding DBNavigator:
, Vcl.DBCtrls
// adding FDMemTable:
, FireDAC.Stan.Intf
, FireDAC.Stan.Option
, FireDAC.Stan.Param
, FireDAC.Stan.Error
, FireDAC.DatS
, FireDAC.Phys.Intf
, FireDAC.DApt.Intf
, FireDAC.Comp.DataSet
, FireDAC.Comp.Client
;

type
  TForm1 = class(TForm)
    pnlTop: TPanel;
    pnlNavigation: TPanel;
    pnlButtons: TPanel;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    DBGrid1: TDBGrid;
    DataSource1: TDataSource;
    DBNavigator1: TDBNavigator;
    FDMemTable1: TFDMemTable;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
  i: word;
begin
  FDMemTable1.DisableControls;
  for i := 1 to 10 do
    begin
```

```
      FDMemTable1.Append;
      FDMemTable1.FieldByName('ID').AsInteger       := i;
      FDMemTable1.FieldByName('Code').AsString    := 'Code'+IntToStr(i);
      FDMemTable1.FieldByName('Created').AsDateTime := Now();
      FDMemTable1.FieldByName('RandNo').AsInteger     := Random(10000);
      FDMemTable1.Post;
    end;
    FDMemTable1.First;
    FDMemTable1.EnableControls;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  FDMemTable1.EmptyDataSet;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  FDMemTable1.IndexFieldNames := 'ID';
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  FDMemTable1.IndexFieldNames := 'RandNo';
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  FDMemTable1.FieldDefs.Add('ID',      ftAutoInc, 0, False);
  FDMemTable1.FieldDefs.Add('Code',  ftString, 10, False);
  FDMemTable1.FieldDefs.Add('Created', ftDateTime,    0, False);
  FDMemTable1.FieldDefs.Add('RandNo',  ftFloat,   0, False);
  FDMemTable1.CreateDataSet;
  DataSource1.DataSet := FDMemTable1;
  DBGrid1.DataSource := DataSource1;
  DBNavigator1.DataSource := DataSource1;
  Self.Caption := 'FireDAC Memory table';
  Self.Position := poScreenCenter;
end;

end.
```

```
unit uMemoryTableCDS;

interface

uses
  Winapi.Windows
, Winapi.Messages
, System.SysUtils
, System.Variants
, System.Classes
, Vcl.Graphics
, Vcl.Controls
, Vcl.Forms
, Vcl.Dialogs
// adding TClientDataSet:
, Data.DB
, Datasnap.DBClient
// adding DBGrid:
, Vcl.Grids
, Vcl.DBGrids
// adding TPanel:
, Vcl.ExtCtrls
// adding TButtons:
, Vcl.StdCtrls
// adding DBNavigator:
, Vcl.DBCtrls
// adding MIDAS:
, MidasLib
;

type
  TForm1 = class(TForm)
    cdsMemTable: TClientDataSet;
    DBGrid1: TDBGrid;
    Panel1: TPanel;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    DataSource1: TDataSource;
    DBNavigator1: TDBNavigator;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
  i: word;
begin
  cdsMemTable.DisableControls;
  for i := 1 to 10 do
    begin
      cdsMemTable.Append;
      cdsMemTable.FieldByName('ID').AsInteger       := i;
      cdsMemTable.FieldByName('Code').AsString      := 'Code'+IntToStr(i);
      cdsMemTable.FieldByName('Created').AsDateTime := Now();
      cdsMemTable.FieldByName('RandNo').AsInteger   := Random(10000);
      cdsMemTable.Post;
    end;
    cdsMemTable.First;
```

```
    cdsMemTable.EnableControls;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  cdsMemTable.EmptyDataSet;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  cdsMemTable.IndexFieldNames := 'ID';
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  cdsMemTable.IndexFieldNames := 'RandNo';
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  cdsMemTable.FieldDefs.Add('ID',      ftAutoInc,  0, False);
  cdsMemTable.FieldDefs.Add('Code',    ftString,  10, False);
  cdsMemTable.FieldDefs.Add('Created', ftDateTime, 0, False);
  cdsMemTable.FieldDefs.Add('RandNo',  ftInteger,  0, False);
  cdsMemTable.CreateDataSet;
  DataSource1.DataSet := cdsMemTable;
  DBGrid1.DataSource := DataSource1;
  DBNavigator1.DataSource := DataSource1;
  Self.Caption := 'ClientDataSet Memory table';
  Self.Position := poScreenCenter;
end;

end.
```