```
unit LCRPopulation;

interface

uses Classes, LCRGlobals, DB;

type

TSafeWaterPopulation = class(TObject)
private
       // Fields
       fBaseYear: Integer;
       fPopData: Array of Array of TPopulationData;
       // Helper fields
       fFilterMinYear: Integer;
       fFilterMaxYear: Integer;
       fFilterRegionIndex: Integer;
       // Methods
       procedure LoadStatsByYearAndRegion(DataSet: TDataSet; Year, RegionIndex:
Integer; var PopData: TPopulationData);
       function LoadStatsByRegion(DataSet: TDataSet; BaseYear, RegionIndex: Integer;
var NoData : boolean): Integer;
       function GetPopulationPct(Year,Region: Integer; Gender: TGender; Age:
Integer): Double;
       function GetPopulationInflator(Year,Region: Integer): Double;
       function GetPopulationInflatorAll(Year,Region: Integer; Age: Integer):
Double;
       function GetBaseYear: Integer;
       procedure Filter(DataSet: TDataSet; var Accept: Boolean);
protected
public
      DebugOn : boolean;

      // Constructors, Destructors
      constructor Create(BaseYear: Integer);

      //data populating methods
      procedure LoadFromDS(DataSet: TDataSet);

      procedure LoadFromFile(aFile : string);
      procedure SaveToFile(aFile : string);
      procedure SaveCSV(aFile : string);
      procedure LoadFromStream(aStream : Tstream);
      procedure SaveToStream(aStream : Tstream);


      // Properties
      property PopulationPct[Year,Region: Integer; Gender: TGender; Age: Integer]:
```

```
Double read GetPopulationPct;
      property PopulationInflator[Year,Region: Integer]: Double read
GetPopulationInflator;
      property PopulationInflatorAll[Year,Region: Integer; Age: Integer]: Double
read GetPopulationInflatorAll;
      property BaseYear: Integer read GetBaseYear;
end;

implementation

uses Math, SysUtils;

// NOTE: Might want to convert from Year as actual Year to Year as Offset.  That is,
//       SafeWaterPop.PopulationPct[2001,0,gMale,a0to4] vs.
//       SafeWaterPop.PopulationPct[0,0,gMale,a0to4]
function TSafeWaterPopulation.GetPopulationPct(Year,Region: Integer; Gender:
TGender; Age: Integer): Double;
var eAge: TAge;
    YearOffset: Integer;
    UseRegion : integer;
begin
     if DebugOn then begin
       Result:=0.00581395;
       exit;
     end;

     UseRegion:=Region;
     YearOffset := Year - fBaseYear;
     eAge       := IntToTAge(Age);
     if ( (YearOffset < 0) or (YearOffset > 49) ) then
        raise Exception.Create('FATAL ERROR: Invalid Year (' + IntToStr(Year) +
').');
     if ( (Region < 0) or (Region > 79) ) then
        raise Exception.Create('FATAL ERROR: Invalid Region (' + IntToStr(Region) +
').');


     //if no pop data exists, use national avg
     if fPopData[UseRegion,YearOffset].TotalPopulation=0 then
       UseRegion:=0;

     if ( eAge <> a85U ) then
        Result := (fPopData[UseRegion,YearOffset].Statistics[Gender,eAge]/5) /
fPopData[UseRegion,YearOffset].TotalPopulation
     else
        Result := fPopData[UseRegion,YearOffset].Statistics[Gender,eAge] /
fPopData[UseRegion,YearOffset].TotalPopulation;
end;
```

```
function TSafeWaterPopulation.GetPopulationInflator(Year,Region: Integer): Double;
var YearOffset: Integer;
    UseRegion : integer;
begin
     if DebugOn then begin
       Result:=1;
       exit;
     end;

     YearOffset := Year - fBaseYear;
     UseRegion:=Region;
     if ( (YearOffset < 0) or (YearOffset > 49) ) then
        raise Exception.Create('FATAL ERROR: Invalid Year (' + IntToStr(Year) +
').');
     if ( (Region < 0) or (Region > 79) ) then
        raise Exception.Create('FATAL ERROR: Invalid Region (' + IntToStr(Region) +
').');

     if fPopData[Region,YearOffset].TotalPopulation=0 then
       UseRegion:=0;

     Result := fPopData[UseRegion,YearOffset].TotalPopulation /
fPopData[UseRegion,0].TotalPopulation;
end;

function TSafeWaterPopulation.GetPopulationInflatorAll(Year,Region: Integer; Age:
Integer): Double;
var YearOffset: Integer;
    UseRegion : integer;
    v1,v2 : double;
begin
     if DebugOn then begin
       Result:=1;
       exit;
     end;

     YearOffset := Year - fBaseYear;
     UseRegion:=Region;
     if ( (YearOffset < 0) or (YearOffset > 49) ) then
        raise Exception.Create('FATAL ERROR: Invalid Year (' + IntToStr(Year) +
').');
     if ( (Region < 0) or (Region > 79) ) then
        raise Exception.Create('FATAL ERROR: Invalid Region (' + IntToStr(Region) +
').');

     if fPopData[Region,YearOffset].TotalPopulation=0 then
       UseRegion:=0;
     v1 :=
fPopData[UseRegion,YearOffset].Statistics[gMale,Tage(Age)]+fPopData[UseRegion,YearOf
fset].Statistics[gFemale,Tage(Age)];
     v2 :=
fPopData[UseRegion,0].Statistics[gMale,Tage(Age)]+fPopData[UseRegion,0].Statistics[g
Female,Tage(Age)];

     Result := v1 / v2;
end;



constructor TSafeWaterPopulation.Create(BaseYear: Integer);
var i,j: Integer;
begin
     // Initialize members.  stfips becomes index
     SetLength(fPopData, 80);
     for i := 0 to High(fPopData) do
     begin
          SetLength(fPopData[i], 50);
          for j:=0 to 49 do
            fillchar(fPopData[i,j],sizeof(fPopData[i,j]),0);
     end;
     fBaseYear := BaseYear;
end;


procedure TSafeWaterPopulation.LoadFromFile(aFile : string);
var InStrm : TFileStream;
begin
  InStrm := TFileStream.Create(aFile, fmOpenRead);
  LoadFromStream(InStrm);
  InStrm.Free;
end;

procedure TSafeWaterPopulation.SaveCSV(aFile : string);
var T : TextFile;
    CurYear,i,g,a : integer;
begin
  assignfile(T,aFile);
  rewrite(T);
  for i := 0 to 79 do begin
    write(T,inttostr(i),',');
    for CurYear:=0 to 49 do begin
      write(T,floattostr(fPopdata[i,CurYear].Statistics[gMale,a10to14]),',');
    end;
    writeln(t);
  end;
  closefile(T);
end;


procedure TSafeWaterPopulation.SaveToFile(aFile : string);
var OutStrm : TFileStream;
begin
  OutStrm := TFileStream.Create(aFile, fmCreate);
  SaveToStream(OutStrm);
  OutStrm.Free;
end;

procedure TSafeWaterPopulation.SaveToStream(aStream : TStream);
var CurYear,i : Integer;
begin
  for CurYear:=0 to 49 do begin
    for i := 0 to 79 do begin
      aStream.Write(fPopData[i,CurYear],SizeOf(fPopData[i,CurYear]));
    end;
  end;
end;

procedure TSafeWaterPopulation.LoadFromStream(aStream : TStream);
var CurYear,i : Integer;
begin
  for CurYear:=0 to 49 do begin
    for i := 0 to 79 do begin
      aStream.Read(fPopData[i,CurYear],SizeOf(fPopData[i,CurYear]));
    end;
  end;
end;




procedure TSafeWaterPopulation.LoadFromDS(DataSet: TDataSet);
var MaxYear,CurYear,i: Integer;
    MaxYearStats,PenultimateYearStats: TPopulationData;
    g: TGender;
    a: TAge;
    NoData : boolean;
    SubPop,Adj : double;
begin
  if DebugOn then exit;

  for i := 1 to High(fPopData) do
  begin
```

```
    // Load all years for which Woods and Poole data is available.
    MaxYear := LoadStatsByRegion(DataSet, BaseYear, i, NoData);
    if NoData then continue;

    // Load MaxYearStats, PenultimateYearStats
    LoadStatsByYearAndRegion(DataSet, MaxYear, i, MaxYearStats);
    LoadStatsByYearAndRegion(DataSet, MaxYear-1, i, PenultimateYearStats);

    // Interpolate statistics for years where Woods and Poole data is not available.
    if ( MaxYear < BaseYear+49 ) then
    begin
      for CurYear := MaxYear+1 to BaseYear+49 do
      begin
        SubPop:=0;
        for g := gMale to gFemale do
        begin
          for a := a0to4 to a85U do
          begin

            if ( PenultimateYearStats.Statistics[g,a] <> 0 ) then
              fPopData[i][CurYear-BaseYear].Statistics[g,a] :=
Trunc(MaxYearStats.Statistics[g,a] *
                                      IntPower(MaxYearStats.Statistics[g,a]/
                                      PenultimateYearStats.Statistics[g,a],
                                      CurYear-MaxYear))
            else
              fPopData[i][CurYear-BaseYear].Statistics[g,a] := 0;

            SubPop:=SubPop+fPopData[i][CurYear-BaseYear].Statistics[g,a];
          end;
        end;

        if ( PenultimateYearStats.TotalPopulation <> 0 ) then
          fPopData[i][CurYear-BaseYear].TotalPopulation :=
Trunc(MaxYearStats.TotalPopulation *
                                     IntPower(MaxYearStats.TotalPopulation/
                                     PenultimateYearStats.TotalPopulation,
                                     CurYear-MaxYear))
        else
          fPopData[i][CurYear-BaseYear].TotalPopulation := 0;

        if fPopData[i][CurYear-BaseYear].TotalPopulation<>SubPop then begin
          //renormalize pops;
          Adj:=fPopData[i][CurYear-BaseYear].TotalPopulation / SubPop ;
          for g := gMale to gFemale do
          begin
            for a := a0to4 to a85U do
            begin
```

```
              fPopData[i][CurYear-BaseYear].Statistics[g,a] :=
                fPopData[i][CurYear-BaseYear].Statistics[g,a] * Adj;
            end;
          end;
        end;
      end;
    end;

   //add to total overall population
    for CurYear:=0 to 49 do begin

fPopData[0,CurYear].TotalPopulation:=fPopData[0,CurYear].TotalPopulation+fPopData[i,
CurYear].TotalPopulation;
      for g := gMale to gFemale do begin
        for a := a0to4 to a85U do begin
          fPopData[0,CurYear].Statistics[g,a]:=
              fPopData[0,CurYear].Statistics[g,a]+
              fPopData[i,CurYear].Statistics[g,a]
        end;
      end;
    end;

  end; {end region loop}
end;

function TSafeWaterPopulation.LoadStatsByRegion(DataSet: TDataSet; BaseYear,
RegionIndex: Integer; var NoData : boolean): Integer;
var FieldName: string;
    CurYear: Integer;
    g: TGender;
    a: TAge;
begin
     Result := BaseYear;

     DataSet.Active   := false;
     DataSet.Filtered := false;
     fFilterMinYear := BaseYear;
     fFilterMaxYear := MaxInt;
     fFilterRegionIndex := RegionIndex;
     DataSet.OnFilterRecord := Filter;
     DataSet.Filtered := true;
     DataSet.Active   := true;

     DataSet.First;
     Nodata:=True;
     while (not DataSet.Eof) do
     begin
          NoData:=False;
          CurYear := DataSet.FieldByName('Year').AsInteger;
          Result  := Max(Result, CurYear);
          CurYear := CurYear - BaseYear;

          for g := gMale to gFemale do
          begin
               for a := a0to4 to a85U do
               begin
                    FieldName := TGenderToStr(g) + TAgeToStr(a);
                    fPopData[RegionIndex,CurYear].Statistics[g,a] :=
fPopData[RegionIndex,CurYear].Statistics[g,a] +

DataSet.FieldByName(FieldName).AsInteger;
                    fPopData[RegionIndex,CurYear].TotalPopulation :=
fPopData[RegionIndex,CurYear].TotalPopulation +

DataSet.FieldByName(FieldName).AsInteger;
               end;
          end;
          DataSet.Next;
     end;
end;

procedure TSafeWaterPopulation.LoadStatsByYearAndRegion(DataSet: TDataSet; Year,
RegionIndex: Integer; var PopData: TPopulationData);
var FieldName: string;
    g: TGender;
    a: TAge;
begin
     FillChar(PopData, sizeof(PopData), 0);

     DataSet.Active   := false;
     DataSet.Filtered := false;
     fFilterMinYear := Year;
     fFilterMaxYear := Year;
     fFilterRegionIndex := RegionIndex;
     DataSet.OnFilterRecord := Filter;
     DataSet.Filtered := true;
     DataSet.Active   := true;

     DataSet.First;
     while (not DataSet.Eof) do
     begin
          for g := gMale to gFemale do
          begin
               for a := a0to4 to a85U do
               begin
                    FieldName := TGenderToStr(g) + TAgeToStr(a);
                    PopData.Statistics[g][a] := PopData.Statistics[g][a] + 

DataSet.FieldByName(FieldName).AsInteger;
                    PopData.TotalPopulation  := PopData.TotalPopulation + 

DataSet.FieldByName(FieldName).AsInteger;
               end;
          end;
          DataSet.Next;
     end;
end;

function TSafeWaterPopulation.GetBaseYear: Integer;
begin
     Result := fBaseYear;
end;

procedure TSafeWaterPopulation.Filter(DataSet: TDataSet; var Accept: Boolean);
var Year: Integer;
    State: Integer;
begin
     Year  := DataSet.FieldByName('Year').AsInteger;
     State := DataSet.FieldByName('State').AsInteger;
     Accept := (fFilterMinYear <= Year) and (Year <= fFilterMaxYear) and 
(State=fFilterRegionIndex);
end;

end.
```