using DALL;

using System;

using System.Collections.Generic;

using System.Linq;

namespace BLL.Logic

{

public static class TravelLogic

{

#region declare

static RavKavEntities db = new RavKavEntities();

static List<Travel> travelsById = new List<Travel>();

static List<Travel> travelsByDate = new List<Travel>();

static Contract currentContract;

static List<Area> areaToCurrentContractTemp;

//Dictionary of contracts, with their own areas

static IDictionary<ContractInformation, List<Area>> contractUsed;

static IDictionary<Travel, int> travelUsed = new Dictionary<Travel, int>();

static List<Contract> contracts = new List<Contract>();

static string type;

// static List<Contract> contractsByDate = new List<Contract>();

//contracts.Where(x => x.AreaToContracts.Any(m => m.Area.Travels.Any(f => (f.userID == id && f.date.Year == date.Year && f.date.Month == date.Month && f.date.Day == i)))).ToList()

static List<CalculateResulte> calculateResultes = new List<CalculateResulte>();

static int conid = 0;

static double difference = 0;

#endregion

public static List<CalculateResulte> CalaulateThePayment(int id, DateTime date)

{

GetTravelsAndContractsByIdAndMonth(id, date);

//free month

type = "freeMounth";

ContractBase();

ContractExtention();

//free day

//Sending travels by day and appropriate contracts for that day

type = "freeDay";

for (int i = 0; i < DateTime.DaysInMonth(date.Year, date.Month); i++)

{

travelsByDate = travelsById.Where(x => x.date.Day == i).ToList();

ContractBase();

ContractExtention();

}

List<Travel> t = new List<Travel>();

foreach (var travel in travelsById)

{

if (!travelUsed.ContainsKey(travel))

t.Add(travel);

}

CalculateResulte c = new CalculateResulte(contracts.Where(x => x.id == 0).FirstOrDefault(), t, false);

calculateResultes.Add(c);

return calculateResultes;

}

public static void GetTravelsAndContractsByIdAndMonth(int id, DateTime date)

{

//The travels for a particular user

travelsById = db.Travels.Where(x => x.userID == id &&

x.date.Year == date.Year &&

x.date.Month == date.Month)

.OrderBy(x => x.price).ThenBy(x => x.areaID).ToList();

//get all contracts appropriate to the user's travel

contracts = db.Contracts.Where(x => x.AreaToContracts.Any(m => m.Area.Travels.Any(f =>

(f.userID == id && f.date.Year == date.Year && f.date.Month == date.Month)))).ToList();

}

//find base contract

//the rule of base contract is:

//contract who as Travel back and forth

//With at least one more internal trip

public static void ContractBase()

{

contractUsed = new Dictionary<ContractInformation, List<Area>>();

//Dictionary of used travels

double price = 0;

List<Travel> travelsToCurrentContract;

//Go over the travel list for the user

for (int i = 0; i < travelsByDate.Count; i++)

{

if (travelUsed.ContainsKey(travelsByDate[i]))

continue;

price = 0;

//Finding the right and cheapest contract

currentContract = travelsByDate[i].Area.AreaToContracts.OrderBy(x => (type == "freeDay" ? x.Contract.freeDay : x.Contract.freeMounth)).FirstOrDefault().Contract;

//Pulling out all travel appropriate to the contract

travelsToCurrentContract = travelsByDate.Where(x => x.Area.AreaToContracts.Any(m => m.contractID == currentContract.id)).ToList();

foreach (var item in travelsToCurrentContract)

{

//Check if some of the travels have already been realized in other contracts if not sum them price

if (!travelUsed.ContainsKey(item))

price += item.price;

}

if ((type == "freeDay" ? currentContract.freeDay : currentContract.freeMounth) <= price)

{

//add the current contract to dictionary contractUsed

// travelsByDay.Where(x=>x.Area

// db.Areas.Where(x => x.AreaToContracts.Any(m => m.contractID == currentContract.id)).ToList());

areaToCurrentContractTemp = new List<Area>();

//Add the travels to the travels list that for them a contract has been found

foreach (var item in travelsToCurrentContract)

{

if (!travelUsed.ContainsKey(item))

{

areaToCurrentContractTemp.Add(item.Area);

travelUsed.Add(item, currentContract.id);

}

}

contractUsed.Add(new ContractInformation(currentContract.id,

(type == "freeDay" ? currentContract.freeDay : currentContract.freeMounth), true), areaToCurrentContractTemp);

}

}

//add signal travels to contract used dictionary

foreach (var travel in travelsByDate)

{

if (!travelUsed.ContainsKey(travel))

contractUsed.Add(new ContractInformation(travel.id, travel.price, false), new List<Area>(travel.areaID));

}

}

public static void ContractExtention()

{

bool b = func();

while (b)

{

Contract con1 = contracts.Where(y => y.id == conid).FirstOrDefault();

ContractInformation con = new ContractInformation(conid, (type == "freeDay" ? con1.freeDay : con1.freeMounth), true);

foreach (var c in contractUsed)

{

if (contracts.Select(x => x.AreaToContracts.Join(c.Value, AreaToCon => AreaToCon.areaID, itemArea => itemArea.id, (AreaToCon, itemArea) => new { AreaToCon, itemArea }).Where(y => y.AreaToCon.contractID == conid)).Any())

{//create list of areas thet include in the extation contract

foreach (var area in c.Value)

{

areaToCurrentContractTemp.Add(area);

}

//adding a single travel that included in the extantion contract to travelUsed

if (c.Key.isContract == false)

travelUsed.Add(travelsById.Where(x => x.id == c.Key.idContractOrTravel).FirstOrDefault(), conid);

//remove contracts and single travels that in the the extantion contract

contractUsed.Remove(c);

foreach (var t in travelUsed)

{ //changing the contract code for travels that were included in the contract that extaned

if (t.Value == conid)

travelUsed[t.Key] = conid;

}

}

}

//add extation contract to contractUsed

contractUsed.Add(con, areaToCurrentContractTemp);

b = func();

}

string day = travelsByDate.Count() == 0 ? null : travelsByDate[0].date.Day.ToString();

//create a travel list for each contract

foreach (var contract in contractUsed)

{

if (contract.Key.isContract == false)

continue;

List<Travel> t = new List<Travel>();

if (type == "freeMounth")

{

foreach (var travel in travelUsed)

{

if (travel.Value == contract.Key.id)

t.Add(travel.Key);

}

}

else

{

foreach (var travel in travelUsed)

{

if (travel.Value == contract.Key.id && travel.Key.date.Day.ToString() == day)

t.Add(travel.Key);

}

}

CalculateResulte conresult = new CalculateResulte(contracts.Where(x => x.id == contract.Key.id).FirstOrDefault(), t, (type == "freeDay" ? true : false));

calculateResultes.Add(conresult);

}

}

public static Contract FindContractExtentionOfTwoSmallContracts(int indexI, int indexJ)

{

List<Area> areaI1 = contractUsed.ElementAt(indexI).Value;

List<Area> areaI2 = contractUsed.ElementAt(indexJ).Value;

//join with 3 tables to find extention contract of two contracts

var extntionContract = contracts.Select(x => x.AreaToContracts.Join

(areaI1, AreaToCon1 => AreaToCon1.areaID, AreI1 => AreI1.id, (AreaToCon1, AreI1) => new { AreaToCon1, AreI1 }).Join

(areaI2, AreaToCon2 => AreaToCon2.AreI1.id, AreI2 => AreI2.id, (AreaToCon2, AreI2) => new { AreaToCon2, AreI2 })

.OrderBy(f => (type == "freeDay" ? f.AreaToCon2.AreaToCon1.Contract.freeDay : f.AreaToCon2.AreaToCon1.Contract.freeMounth))

.FirstOrDefault().AreaToCon2.AreaToCon1.Contract).FirstOrDefault();

return extntionContract;

}

//public static bool AddTravelsToTheExtentionContract(Contract extntionContract)

// {

// int conid = 0;

// double price = 0;

// bool response = false;

// List<Area> areaToCurrentContractTemp = new List<Area>();

// double sumOfTravelPrice = 0;

// // IDictionary<int, double> extentionTemp = new Dictionary<int, double>();

// //check if the extention contract is cheapest

// if ((type == "freeDay" ? extntionContract.freeDay : extntionContract.freeMounth) <= sumOfTravelPrice && sumOfTravelPrice - (type == "freeDay" ? extntionContract.freeDay : extntionContract.freeMounth) > price)

// {

// conid = extntionContract.id;

// price = (type == "freeDay" ? extntionContract.freeDay : extntionContract.freeMounth);

// // extentionTemp.Add(extntionContract.id, sumOfTravelPrice - (type == "freeDay" ? extntionContract.freeDay : extntionContract.freeMounth));

// #region

// /\* foreach (var item in contractUsed)

// {

// if (contracts.Select(x => x.AreaToContracts.Join

// (item.Value, AreaToCon => AreaToCon.areaID, itemArea => itemArea.id, (AreaToCon, itemArea) => new { AreaToCon, itemArea })

// .Where(y => y.AreaToCon.contractID == extntionContract.id)

// ).Any())

// {

// //add the areas of all contracts that include in the extention contract

// foreach (var area in item.Value)

// {

// areaToCurrentContractTemp.Add(area);

// }

// //remove all contracts that include in the extention contract

// contractUsed.Remove(item);

// }

// }\*/

// #endregion

// //add extention contract to contractUsed

// contractUsed.Add(new ContractInformation(extntionContract.id, (type == "freeDay" ? extntionContract.freeDay : extntionContract.freeMounth)

// , true), areaToCurrentContractTemp);

// response = true;

// }

// //return true if ther is extention contract

// return response;

// }

public static bool func()

{

conid = 0;

difference = 0;

Contract extntionContract;

double sumOfTravelPrice = 0;

for (int i = 0; i < contractUsed.Count() - 1; i++)

{

for (int j = i + 1; j < contractUsed.Count(); j++)

{

//send two contract to check if there is extantion contract for them

extntionContract = FindContractExtentionOfTwoSmallContracts(i, j);

if (extntionContract != null)

{

foreach (var item in contractUsed)

{

//sum price of all contracts that include in the extention contract

if (contracts.Select(x => x.AreaToContracts.Join

(item.Value, AreaToCon => AreaToCon.areaID, itemArea => itemArea.id, (AreaToCon, itemArea) => new { AreaToCon, itemArea })

.Where(y => y.AreaToCon.contractID == extntionContract.id)).Any())

{

sumOfTravelPrice += item.Key.price;

}

}

///check if the extention contract is cheapest

if ((type == "freeDay" ? extntionContract.freeDay : extntionContract.freeMounth) <= sumOfTravelPrice && sumOfTravelPrice - (type == "freeDay" ? extntionContract.freeDay : extntionContract.freeMounth) > difference)

{

conid = extntionContract.id;

difference = (type == "freeDay" ? extntionContract.freeDay : extntionContract.freeMounth);

}

}

}

}

if (conid == 0 && difference == 0)

return false;

else

return true;

}

}

}