**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з комп’ютерного практикуму №1 з дисципліни

«Основи розробки програмного забезпечення на платформі Microsoft.NET»

**«LINQ to Objects»**

**Виконав**

(шифр, прізвище, ім'я, по батькові)

*Голяка Д.В.*

**Перевірила**

(прізвище, ім'я, по батькові)

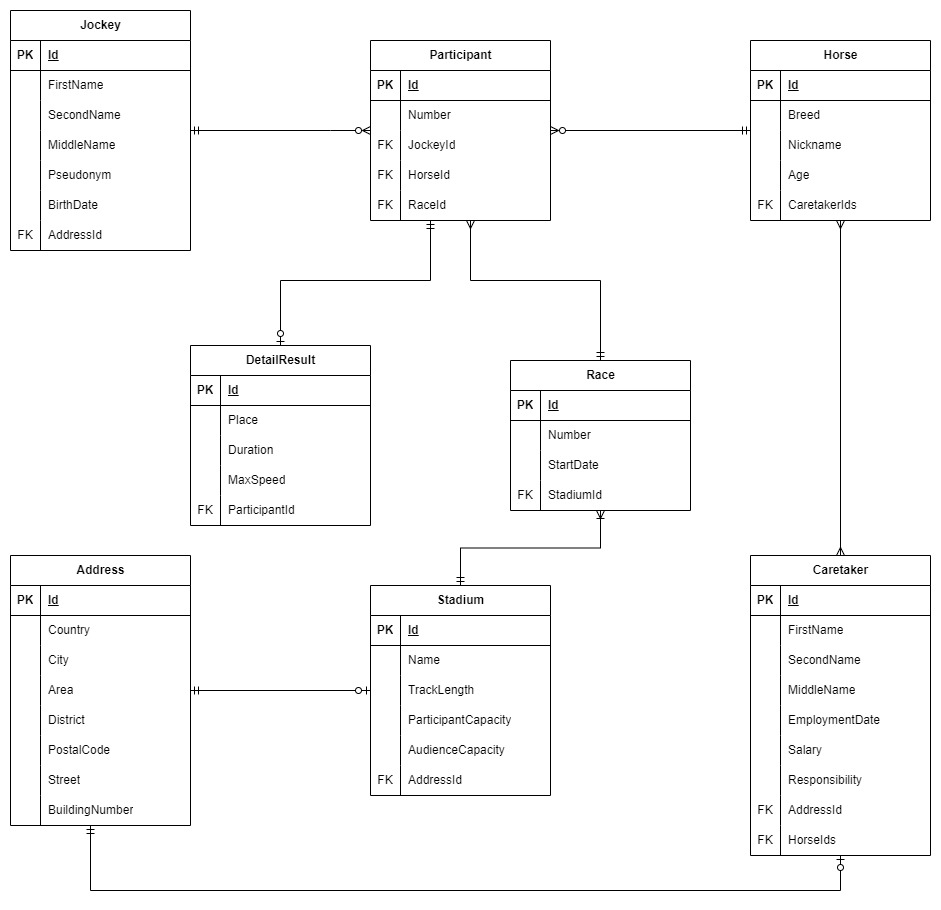
*Ліщук К.І.*

**Варіант 16**

**Умова**

Розробити структуру даних для зберігання інформації про скачки та проведені заїзди. Жокей характеризується псевдонімом, прізвищем, ім’ям, по батькові, датою народження. Коні, які беруть участь в перегонах, характеризуються породою, кличкою і віком. Заїзди проводяться в певну дату і час, мають певний номер (в один день може бути проведено кілька заїздів). Учасники заїзду характеризуються номером і зайнятим місцем в заїзді. Під учасником розуміється жокей і кінь, на якому він буде брати участь в заїзді

**ER-діаграма**

****

**Опис архітектури**

Проєкт був декомпозований на наступні каталоги для зручності роботи:

DataCollector – представляє контекст даних, клас, в якому агреговано колекції даниз;

Entities – представляє набір сутностей (адреси, жокеї, коні та інші)

DTO – представляє об’єкти для отримання результату та зручності передачі.

ER-diagram – представляє ER-діаграму заданого предметного середовища

Execution – представляє набір класів для виконання додатку

Inserters – представляє класи для вставки даних у DataContext

Service – допоміжні класи, що відповідають за логіку роботи програми

Більш детально із архітектурою додатку можна ознайомитися за посиланням: [GitHub](https://github.com/DmytroHoliaka/labs-dotnet)

**Запити**

/// <summary>

/// Query 1. (Method syntax)

/// Returns a summary of all horses in descending order of age

/// </summary>

public static List<HorseShortInfo> GetHorsesByDescendingAge(List<Horse> horses)

{

List<HorseShortInfo> res = horses

.OrderByDescending(h => h.Age)

.Select(h => new HorseShortInfo()

{

Id = h.Id,

Nickname = h.Nickname,

Age = h.Age

})

.ToList();

return res;

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 2. (Method syntax)

/// Returns addresses from the United States with an odd house number

/// </summary>

public static List<AddressShortInfo> GetFilteredAddresses(List<Address> addresses)

{

List<AddressShortInfo> res = addresses

.Where(a => a.Country == "USA" && a.BuildingNumber % 2 == 1)

.Select(a => new AddressShortInfo()

{

Id = a.Id,

Country = a.Country,

BuildingNumber = a.BuildingNumber,

})

.ToList();

return res;

}

Изображение выглядит как текст, Шрифт, снимок экрана, Графика

Автоматически созданное описание

/// <summary>

/// Query 3. (Method syntax)

/// Returns jockeys who live in the specified country

/// </summary>

public static List<JockeyCountryInfo> GetJockeysInCountry(List<Jockey> jockeys, List<Address> addresses, string? country)

{

if (country is null)

{

Console.WriteLine("Country cannot be null");

return [];

}

List<JockeyCountryInfo> res = jockeys

.Join(addresses,

j => j.AddressId,

a => a.Id,

(j, a) => new JockeyCountryInfo()

{

Id = j.Id,

FullName = j.FirstName + ' ' + j.MiddleName + ' ' + j.SecondName,

Country = a.Country

})

.Where(g => g.Country == country)

.ToList();

return res;

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 4. (Query syntax)

/// Returns all caretakers who care for more than one horse

/// </summary>

public static List<CaretakerHorseCountInfo> GetCaretakerWithMoreThanOneHorse(List<Caretaker> caretakers)

{

IEnumerable<CaretakerHorseCountInfo> res =

from c in caretakers

let horseCount = c.HorseIds!.Count()

where horseCount > 1

select new CaretakerHorseCountInfo()

{

Id = c.Id,

FullName = c.FirstName + ' ' + c.MiddleName + ' ' + c.SecondName,

Salary = c.Salary,

HorseCount = horseCount

};

return res.ToList();

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 5. (Query syntax)

/// Returns horses that have won more than once

/// </summary>

public static List<HorseWinsInfo> GetHorsesWithMultipleWins(List<Horse> horses, List<Participant> participants, List<DetailResult> results)

{

IEnumerable<HorseWinsInfo> res =

from h in horses

join p in participants on h.Id equals p.HorseId

join r in results on p.Id equals r.ParticipantId

group r by new { h.Id, h.Nickname } into grouped

let winningCount = grouped.Count(r => r.Place == 1)

where winningCount > 1

select new HorseWinsInfo()

{

Id = grouped.Key.Id,

Nickname = grouped.Key.Nickname,

WinningCount = winningCount,

};

return res.ToList();

}

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

/// <summary>

/// Query 6. (Query syntax)

/// Returns horses that have participated in races

/// </summary>

public static List<HorseShortInfo> GetHorsesInRacing(List<Horse> horses, List<Participant> participants)

{

IEnumerable<HorseShortInfo> res =

from h in horses

join p in participants on h.Id equals p.HorseId

group h by new { h.Id, h.Nickname, h.Age } into horseGroup

select new HorseShortInfo()

{

Id = horseGroup.Key.Id,

Nickname = horseGroup.Key.Nickname,

Age = horseGroup.Key.Age,

};

return res.ToList();

}

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

/// <summary>

/// Query 7. (Method syntax)

/// Return horses that have not participated in a race

/// </summary>

public static List<HorseShortInfo> GetHorsesWithoutRacing(List<Horse> horses, List<Participant> participants)

{

List<HorseShortInfo> res = horses

.Where(h => participants.Any(p => p.HorseId == h.Id) == false)

.Select(h => new HorseShortInfo()

{

Id = h.Id,

Nickname = h.Nickname,

Age = h.Age

})

.ToList();

return res;

}

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

/// <summary>

/// Query 8. (Method syntax)

/// Returns jockeys older than the number of years specified by the user

/// </summary>

public static List<JockeyAgeInfo> GetJockeysOlderThan(List<Jockey> jockeys, int age)

{

List<JockeyAgeInfo> res = jockeys

.Where(j => (DateTime.Now - j.BirthDate) > TimeSpan.FromDays(age \* 365))

.Select(j => new JockeyAgeInfo()

{

Id = j.Id,

FullName = j.FirstName + ' ' + j.MiddleName + ' ' + j.SecondName,

Age = Math.Round((DateTime.Now - j.BirthDate).Days / 365.25, 2),

})

.ToList();

return res;

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 9. (Method syntax)

/// Returns a horse with a record top speed

/// </summary>

public static HorseMaxSpeedInfo? GetBestSpeedHorse(List<Horse> horses, List<Participant> participants, List<DetailResult> results)

{

HorseMaxSpeedInfo? horse = horses

.Join(participants,

h => h.Id,

p => p.HorseId,

(h, p) => new

{

h.Id,

h.Breed,

h.Nickname,

ParticipantId = p.Id

})

.Join(results,

p => p.ParticipantId,

r => r.ParticipantId,

(p, r) => new HorseMaxSpeedInfo()

{

Id = p.Id,

Breed = p.Breed,

Nickname = p.Nickname,

MaxSpeed = r.MaxSpeed,

})

.MaxBy(g => g.MaxSpeed);

return horse;

}

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

/// <summary>

/// Query 10. (Method syntax)

/// Returns all jockeys and horses by combining the corresponding pairs

/// </summary>

public static List<JockeyHorseInfo> GetJockeysAndHorses(List<Jockey> jockeys, List<Horse> horses, List<Participant> participants)

{

IEnumerable<JockeyHorseInfo> jockeysHorsesLeftJoin = jockeys

.GroupJoin(participants,

j => j.Id,

p => p.JockeyId,

(j, participantGroup) => new

{

JockeyId = j.Id,

FullName = j.FirstName + ' ' + j.MiddleName + ' ' + j.SecondName,

ParticipantGroup = participantGroup,

})

.SelectMany(p => p.ParticipantGroup.DefaultIfEmpty(),

(p, participant) => new

{

p.FullName,

HorseId = participant?.HorseId ?? -1

})

.GroupJoin(horses,

p => p.HorseId,

h => h.Id,

(p, horseGroup) => new

{

p.FullName,

HorseGroup = horseGroup

})

.SelectMany(g => g.HorseGroup.DefaultIfEmpty(),

(p, horse) => new JockeyHorseInfo()

{

FullName = p.FullName,

HorseNickname = horse?.Nickname ?? "<without any horse>"

})

.Distinct();

IEnumerable<JockeyHorseInfo> jockeysHorsesRightJoin = horses

.GroupJoin(participants,

h => h.Id,

p => p.HorseId,

(h, participantGroup) => new

{

HorseNickname = h.Nickname,

ParticipantGroup = participantGroup

})

.SelectMany(p => p.ParticipantGroup.DefaultIfEmpty(),

(p, participant) => new

{

JockeyId = participant?.JockeyId ?? -1,

p.HorseNickname

})

.GroupJoin(jockeys,

g => g.JockeyId,

j => j.Id,

(g, groupJockey) => new

{

g.JockeyId,

g.HorseNickname,

GroupJockey = groupJockey,

})

.SelectMany(p => p.GroupJockey.DefaultIfEmpty(),

(p, jockey) => new JockeyHorseInfo()

{

FullName = (jockey is not null)

? jockey?.FirstName + ' ' + jockey?.MiddleName + ' ' + jockey?.SecondName

: "<without any jockey>",

HorseNickname = p.HorseNickname,

})

.Distinct();

List<JockeyHorseInfo> res = jockeysHorsesLeftJoin.Union(jockeysHorsesRightJoin!).ToList();

return res;

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 11. (Method syntax)

/// Return the stadium with the longest track

/// </summary>

public static StadiumMaxTrackInfo? GetStadiumWithMaxTrackLength(List<Stadium> stadiums)

{

StadiumMaxTrackInfo? res = stadiums

.Select(s => new StadiumMaxTrackInfo()

{

Id = s.Id,

Name = s.Name,

TrackLength = s.TrackLength

})

.MaxBy(s => s.TrackLength);

return res;

}

Изображение выглядит как текст, Шрифт, снимок экрана, Графика

Автоматически созданное описание

/// <summary>

/// Query 12. (Query syntax)

/// Return the caretaker who lives in the specified country and earns more than the specified amount of money

/// </summary>

public static List<CaretakerCountryInfo> GetCaretakersInCountryWithSalaryAbove(List<Caretaker> caretakers, List<Address> addresses, string? country, int salary)

{

if (country is null)

{

Console.WriteLine("Country cannot be null");

return [];

}

IEnumerable<CaretakerCountryInfo> res =

from c in caretakers

where c.Salary > salary

join a in addresses on c.AddressId equals a.Id

where a.Country == country

select new CaretakerCountryInfo()

{

Id = c.Id,

FullName = c.FirstName + ' ' + c.MiddleName + ' ' + c.SecondName,

Salary = c.Salary,

Country = a.Country,

};

return res.ToList();

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 13. (Method syntax)

/// Returns races that took place at a stadium in the specified country

/// </summary>

public static List<RaceCountryInfo> GetRacesInCountry(List<Race> races, List<Stadium> stadiums, List<Address> addresses, string? country)

{

if (country is null)

{

Console.WriteLine("Country cannot be null");

return [];

}

List<RaceCountryInfo> res = races

.Join(stadiums,

r => r.StadiumId,

s => s.Id,

(r, s) => new

{

r.Id,

r.Number,

s.AddressId,

})

.Join(addresses,

at => at.AddressId,

a => a.Id,

(at, a) => new RaceCountryInfo()

{

Id = at.Id,

Number = at.Number,

Country = a.Country,

})

.Where(at => at.Country == country)

.ToList();

return res;

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

// <summary>

/// Query 14. (Query syntax)

/// Returns horses that have a trainer

/// </summary>

public static List<HorseTrainerInfo> GetHorsesWithTrainer(List<Horse> horses, List<Caretaker> caretakers)

{

IEnumerable<HorseTrainerInfo> res =

from h in horses

from caretakerId in h.CaretakerIds ?? [-1]

join c in caretakers on caretakerId equals c.Id

where c.Responsibility == "Horse Training"

select new HorseTrainerInfo()

{

HorseId = h.Id,

HorseNickname = h.Nickname,

CaretakerType = c.Responsibility

};

return res.ToList();

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 15. (Query syntax)

/// Returns jockeys whose name ends in a vowel in descending order of age

/// </summary>

public static List<JockeyAgeInfo> GetVowelNamedJockeysByAgeDesc(List<Jockey> jockeys)

{

char[] vowes = ['a', 'e', 'i', 'o', 'u', 'y'];

IEnumerable<JockeyAgeInfo> res =

from j in jockeys

let lastLetter = j.FirstName?.Last() ?? '\0'

let age = (DateTime.Now - j.BirthDate).Days

where vowes.Contains(lastLetter)

orderby age descending

select new JockeyAgeInfo()

{

Id = j.Id,

FullName = j.FirstName + ' ' + j.MiddleName + ' ' + j.SecondName,

Age = age / 365.25,

};

return res.ToList();

}

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

/// <summary>

/// Query 16. (Query syntax)

/// Returns caretakers who look after the specified breed of horse

/// </summary>

public static List<CaretakerHorseBreedInfo> GetCaretakersForBreed(List<Caretaker> caretakers, List<Horse> horses, string? breed)

{

if (breed is null)

{

Console.WriteLine("Breed cannot be null");

return [];

}

IEnumerable<CaretakerHorseBreedInfo> res =

from c in caretakers

from horseId in c.HorseIds ?? [-1]

join h in horses on horseId equals h.Id

where h.Breed == breed

group new { h, c } by new { c.Id, c.FirstName, c.MiddleName, c.SecondName, h.Breed } into grouped

select new CaretakerHorseBreedInfo()

{

Id = grouped.Key.Id,

FullName = grouped.Key.FirstName + ' ' + grouped.Key.MiddleName + ' ' + grouped.Key.SecondName,

HorseBreed = grouped.Key.Breed,

};

return res.ToList();

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 17. (Query syntax)

/// Returns the races in which the specified horses participated by nickname

/// </summary>

public static List<RaceHorseNicknameCount> GetRacesByHorseNickname(List<Race> races, List<Participant> participants, List<Horse> horses, string? nickname)

{

if (nickname is null)

{

Console.WriteLine("Nickname cannot be null");

return [];

}

IEnumerable<RaceHorseNicknameCount> res =

from r in races

join p in participants on r.Id equals p.RaceId

join h in horses on p.HorseId equals h.Id

group new { r, h } by new { r.Id, r.Number, h.Nickname } into grouped

where grouped.Any(a => a.h.Nickname == nickname)

select new RaceHorseNicknameCount()

{

Id = grouped.Key.Id,

Number = grouped.Key.Number,

HorseCount = grouped.Count(a => a.h.Nickname == nickname),

};

return res.ToList();

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

/// <summary>

/// Query 18. (Query syntax)

/// Returns the horses in which the jockey with the specified ID participated

/// </summary>

public static List<JockeyHorseInfo> GetHorsesRiddenByJockey(List<Jockey> jockeys, List<Participant> participants, List<Horse> horses, int jockeyId)

{

IEnumerable<JockeyHorseInfo> res =

from j in jockeys

where j.Id == jockeyId

join p in participants on j.Id equals p.JockeyId

join h in horses on p.HorseId equals h.Id

select new JockeyHorseInfo()

{

FullName = j.FirstName + ' ' + j.MiddleName + ' ' + j.SecondName,

HorseNickname = h.Nickname,

};

return res.ToList();

}

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

/// <summary>

/// Query 19. (Method syntax)

/// Returns the jockey who took the first place the most times

/// </summary>

public static JockeyWinsInfo? GetTopWinningJockey(List<Jockey> jockeys, List<Participant> participants, List<DetailResult> results)

{

JockeyWinsInfo? res = jockeys

.Join(participants,

j => j.Id,

p => p.JockeyId,

(j, p) => new

{

JockeyId = j.Id,

JockeyFullName = j.FirstName + ' ' + j.MiddleName + ' ' + j.SecondName,

ParticipantId = p.Id,

})

.Join(results.Where(r => r.Place == 1),

jp => jp.ParticipantId,

r => r.ParticipantId,

(jp, r) => new

{

jp.JockeyId,

jp.JockeyFullName,

r.Place,

})

.GroupBy(jr => new { jr.JockeyId, jr.JockeyFullName },

jr => jr.Place,

(j, p) => new JockeyWinsInfo()

{

JockeyId = j.JockeyId,

FullName = j.JockeyFullName,

WinningCount = p.Count(),

})

.MaxBy(jc => jc.WinningCount);

return res;

}



/// <summary>

/// Query 20. (Query syntax)

/// Returns jockeys who participated in the stadium of the specified country

/// </summary>

public static List<JockeyStadiumCountry> GetJockeysByStadiumCountry(List<Jockey> jockeys, List<Participant> participants, List<Stadium> stadiums,

List<Race> races, List<Address> addresses, string country)

{

if (country is null)

{

Console.WriteLine("Country cannot be null");

return [];

}

IEnumerable<JockeyStadiumCountry> res =

from a in addresses

where a.Country == country

join s in stadiums on a.Id equals s.AddressId

join r in races on s.Id equals r.Id

join p in participants on r.Id equals p.RaceId

join j in jockeys on p.JockeyId equals j.Id

group new { j, a } by new { j.Id, j.FirstName, j.MiddleName, j.SecondName, a.Country } into grouped

select new JockeyStadiumCountry()

{

Id = grouped.Key.Id,

FullName = grouped.Key.FirstName + ' ' + grouped.Key.MiddleName + ' ' + grouped.Key.SecondName,

StadiumCountry = grouped.Key.Country,

};

return res.ToList();

}

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание