TALLINNA TEHNIKAÜLIKOOL

School of Information Technologies

Kerman Saapar IADB184996

Dog Agility Time Measuring System Web App

Distributed systems project

Supervisor: Andres Käver

Tallinn 2020

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Kerman Saapar

[dd.mm.yyyy]

Table of Contents

Introduction

Dog agility lacks a cost effective and innovative time measuring system. Today’s competitions use big screens only for current runner time. Also current time measuring systems cost 5000+€, use long cables and are bulky to carry around. People with bad vision or bad sun angle can’t see those screens. Every person nowadays has a smartphone in his/her pocket. For that reason, there is a possibility to innovate dog agility.

The complete system involves hardware, but in the scope of this project, the author is focusing on the software aspect only.

The purpose of this work is to build a competition/training times application as a distributed system, which would comply with the demands of the course.

To fulfill this work, the author will create an ERD schema, creates a database and builds application layers respectively to the lecturer’s requests and wishes.

The course work consists of the documentation (the given document), ERD schema and application source code.

1. Overview

The system is aimed for hobbyists.

The user can see his/her dog running times from his/her smartphone. Follow his/her favorite dog’s times. Since physical hardware supports time logging from multiple obstacles, there’s a possibility to compare each other’s track times by obstacle.

The user can also see top times by competition/track/dog breed/class.

When there’s an ongoing competition, connected user can see the current run’s current time from his/her smartphone. This replaces big screens.

1.1 Hardware

Briefly about hardware.

1-10 physical gates which can be placed at each obstacle on the obstacle course. Gates use 3 IR sensors for detecting passed object. They work on the same principal as security curtains at big industrial machines.

Every gate is connected to hub via Wi-Fi. Hub is likely Orange Pi or Banana Pi, because they have EMMC ram, not SD card like Raspberry Pi.

Hub is connected to our server every time it’s turned on via 3G or 4G. Connection is needed for validation, if the system subscription is still valid and the money has been payed.

Everything is powered by portable power banks.

1.2 Web interface needs

The main view for non-registered users is participants time’s table. This is similar to WRC rally results table. There you can see every result for ongoing competition. See every checkpoint’s time, is he/she faster than the leader, by how much etc. A big timer pops up, when there is an ongoing run.

Registered user can also bookmark his/her favorite dog/owner, to see their track times and compare it to their own.

Admin can set up new competitions and tracks. Manage results in case of an error and edit them. Manage physical gates queue

Analysis

Soft CRUD

The main idea behind soft CRUD is that we want to keep track of what happened in the past. We don't delete anything really. Imagine a slider, where you can slide current time, what was in the database at that time. How to achieve this?

One way is to create history table. For example, there is Person and PersonHistory tables, every record is copied to PersonHistory table and logged with a timestamp when was the change done and finally update the correct record in the Person table. But that doesn't seem feasible because when you create a second table for every table, then the tables would double and it gets out of hand pretty quickly.

Another way is to use composite keys. Composite key is made of row's "ID" and "Deleted at" field. When a record is deleted, the deleted at field is set. There would be many same ID fields with different deleted at. This means that a record has many versions.

Author found that the most universal way is to use composite keys which is made of "ID" and "Deleted at".

In the demo, there are only soft update operations, because soft delete is already a step in soft update.

1:0-M

The first demo contains 1:0-m situation.

Firstly a database is created.

```
IF db_id('kesaap_hw2') IS NOT NULL BEGIN
 USE master
 DROP DATABASE "kesaap_hw2"
END
GO
CREATE DATABASE "kesaap_hw2"
GO
USE "kesaap_hw2"
GO
```

Second, 2 tables are created, breed and animal which are 1:0-m relationship. The deleted at field is not nullable, the most logical choice is to use a date far in the future, like year 9999. This is done, because it simplifies SELECT queries and is easier to understand.

```
CREATE TABLE Breed (
 Id INT NOT NULL,
 Name VARCHAR(255) NOT NULL,
 CreatedAt DATETIME2 NOT NULL,
 DeletedAt DATETIME2 NOT NULL,
 CONSTRAINT PK_Track PRIMARY KEY (Id, DeletedAt)
)

CREATE TABLE Animal (
 Id INT NOT NULL,
 Name VARCHAR(255) NOT NULL,
 Age INT,
 CreatedAt DATETIME2 NOT NULL,
 DeletedAt DATETIME2 NOT NULL,
 Breed_Id INT NOT NULL,
 Breed_DeletedAt DATETIME2
 CONSTRAINT PK_Animal PRIMARY KEY (Id, DeletedAt),
 CONSTRAINT FK_Animal_Breed FOREIGN KEY (Breed_Id, Breed_DeletedAt) REFERENCES Breed(Id, Delet
edAt)
)
```

Next some test data is added.

```
DECLARE @Time1 DATETIME2
SELECT @Time1 = '2017-07-30'

INSERT INTO Breed (Id, Name, CreatedAt, DeletedAt)
VALUES (1, 'Border Collie', @Time1, '9999-12-31')

INSERT INTO Breed (Id, Name, CreatedAt, DeletedAt)
VALUES (2, 'Jack Russell Terrier', @Time1, '9999-12-31')

INSERT INTO Animal (Id, Name, Age, CreatedAt, DeletedAt, Breed_Id, Breed_DeletedAt)
VALUES (1, 'Yes', 2, @Time1, '9999-12-31', 1, '9999-12-31')

INSERT INTO Animal (Id, Name, Age, CreatedAt, DeletedAt, Breed_Id, Breed_DeletedAt)
VALUES (2, 'Fii', 1, @Time1, '9999-12-31', 1, '9999-12-31')
```

Now the original Breed name “Border Collie” is updated to “Border Collie v2”. First a copy is made of the original record. Then the deleted at date is changed on the copy record and lastly the original record is updated to “Border Collie v2”.

```
DECLARE @Time2 DATETIME2
SELECT @Time2 = '2018-07-30'
```

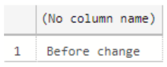
```
DECLARE @OriginalId INT
SELECT @OriginalId = Id FROM Breed WHERE Name LIKE 'Border Collie'

INSERT INTO Breed (Id, Name, CreatedAt, DeletedAt) SELECT Id, Name, CreatedAt, @Time2 FROM Bre
ed WHERE Id = @OriginalId
UPDATE Breed SET CreatedAt = @Time2 WHERE Id = @OriginalId AND DeletedAt = '9999-12-31'
UPDATE Breed SET Name = 'Border Collie v2' WHERE Id = @OriginalId AND DeletedAt = '9999-12-31'
SELECT * FROM BREED
```

The same is done to the Animal named “Yes”. It is renamed to “No”.

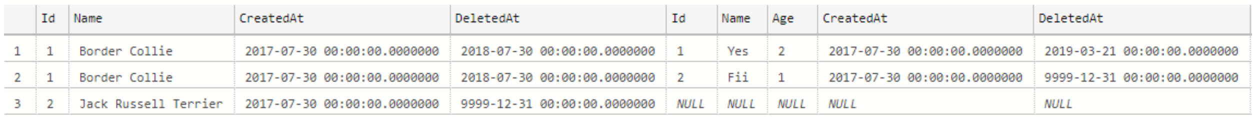
```
DECLARE @Time3 DATETIME2
SELECT @Time3 = '2019-03-21'
DECLARE @OriginalId2 INT
SELECT @OriginalId2 = Id FROM Animal WHERE Name LIKE 'Yes'

INSERT INTO Animal (Id, Name, Age, CreatedAt, DeletedAt, Breed_Id, Breed_DeletedAt)
SELECT Id, Name, Age, CreatedAt, @Time3, Breed_Id, Breed_DeletedAt FROM Animal WHERE Id = @Ori
ginalId2
UPDATE Animal SET CreatedAt = @Time3 WHERE Id = @OriginalId2 AND DeletedAt = '9999-12-31'
UPDATE Animal SET Name = 'No' WHERE Id = @OriginalId2 AND DeletedAt = '9999-12-31'
SELECT * FROM Animal
```

Lastly two SELECT queries is done to show how changing the date gives two different results.

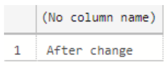
```
DECLARE @TimeNow DATETIME2
SELECT @TimeNow = '2018-03-20'

SELECT 'Before change'
-- Include Animal table with left join based on ID
SELECT * FROM Breed LEFT JOIN Animal ON Breed.Id = Animal.Breed_Id
-- CreatedAt has to be smaller than the checked time
WHERE Breed.CreatedAt <= @TimeNow
-- DeletedAt has to be bigger than the checked time
-- So that the checked time lands between times
AND Breed.DeletedAt > @TimeNow
-- Select animals who are linked to that breed
--and created at and deleted at also have to be in between checked time
AND ((Animal.Id IS NOT NULL
AND Animal.CreatedAt <= @TimeNow
AND Animal.DeletedAt > @TimeNow)
-- OR include entries which dont have any animals linked to them
OR (Animal.Id IS NULL))

DECLARE @TimeNow2 DATETIME2
SELECT @TimeNow2 = '2019-03-22'

SELECT 'After change'
SELECT * FROM Breed LEFT JOIN Animal ON Breed.Id = Animal.Breed_Id
WHERE Breed.CreatedAt <= @TimeNow2
AND Breed.DeletedAt > @TimeNow2
AND ((Animal.Id IS NOT NULL
AND Animal.CreatedAt <= @TimeNow2
AND Animal.DeletedAt > @TimeNow2)
OR (Animal.Id IS NULL))
```



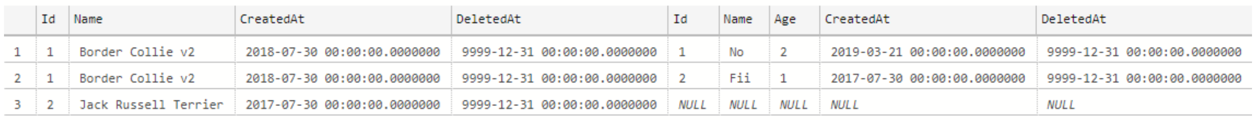
	(No column name)
1	Before change



	Id	Name	CreatedAt	DeletedAt	Id	Name	Age	CreatedAt	DeletedAt
1	1	Border Collie	2017-07-30 00:00:00.0000000	2018-07-30 00:00:00.0000000	1	Yes	2	2017-07-30 00:00:00.0000000	2019-03-21 00:00:00.0000000
2	1	Border Collie	2017-07-30 00:00:00.0000000	2018-07-30 00:00:00.0000000	2	Fii	1	2017-07-30 00:00:00.0000000	9999-12-31 00:00:00.0000000
3	2	Jack Russell Terrier	2017-07-30 00:00:00.0000000	9999-12-31 00:00:00.0000000	NULL	NULL	NULL	NULL	NULL



	(No column name)
1	After change



	Id	Name	CreatedAt	DeletedAt	Id	Name	Age	CreatedAt	DeletedAt
1	1	Border Collie v2	2018-07-30 00:00:00.0000000	9999-12-31 00:00:00.0000000	1	No	2	2019-03-21 00:00:00.0000000	9999-12-31 00:00:00.0000000
2	1	Border Collie v2	2018-07-30 00:00:00.0000000	9999-12-31 00:00:00.0000000	2	Fii	1	2017-07-30 00:00:00.0000000	9999-12-31 00:00:00.0000000
3	2	Jack Russell Terrier	2017-07-30 00:00:00.0000000	9999-12-31 00:00:00.0000000	NULL	NULL	NULL	NULL	NULL

1:0-1

1:0-1 is not possible for soft update, because when you make an update, you would need a copy. This already violates the point of 1:0-1, because then you would have 2 records pointing to the same parent ID which results to 1:0-M relationship.

One way is to create a new parent and new childs and link them all together, but that would create a lot of useless records and just isn’t feasible. Its simpler to create 1:0-M relationship.

Conclusion

Soft CRUD takes a lot of thinking to implement. There is no single right answer how to implement it, because there are many options to achieve this. Also soft CRUD is a double-edged sword. Now you need to take into consideration GDPR, when a user demands that you delete their data. There is some data that you need to keep based on laws and some data that you need to delete.

Repository

The problem

Imagine manipulating data from directly business logic. What happens when there's a need to change the database? Different database has quirks that are mandatory to adapt to. This means that it's needed to rewrite business logic to adapt to the new database. Which means extra work. How can the business logic be unit tested, when the data operations are done directly in the business logic? You must rewrite every data operation also.

Repository pattern

Repository pattern separates data operations from business logic. Business logic knows nothing how the data is inserted into the database and how it's taken from it. This means that its loosely coupled. The data layer is completely abstracted away, which allows to swap out the database management system for some other provider or use completely different type of data persistence layer.

The repository handles with domain objects.

When the business logic doesn't know about data operations, its easy to unit test business logic. Because you can send mocked data straight into business logic.

Data Access Object (DAO) pattern

DAO is much closer to data storage. DAO allows for a simpler way to get data from a storage, hiding ugly queries. Its goal is like of the repository pattern, but it is not as flexible as the repository pattern is. DAO is a class that locates the data. The pattern doesn't restrict you to store data of the same type, so you can easily have a DAO that locates/store related objects.

Difference

Repository is an abstraction of DAO. You may use multiple DAO's inside a repository. A repository deals with domain objects hence higher level. DAO is lower level, closer to the storage, dealing only with data. Its easier to unit test a repository. You may also implement caching into repository

Project implementation

In author’s project, he will use the repository pattern to separate business and database logic. Because its more flexible. This also allows to reuse code in any upcoming projects that may come.

Currently there is a base repository which implements all basic CRUD and every specific repository derive from base repository. This means that every specific repository could add specific methods for that repository.

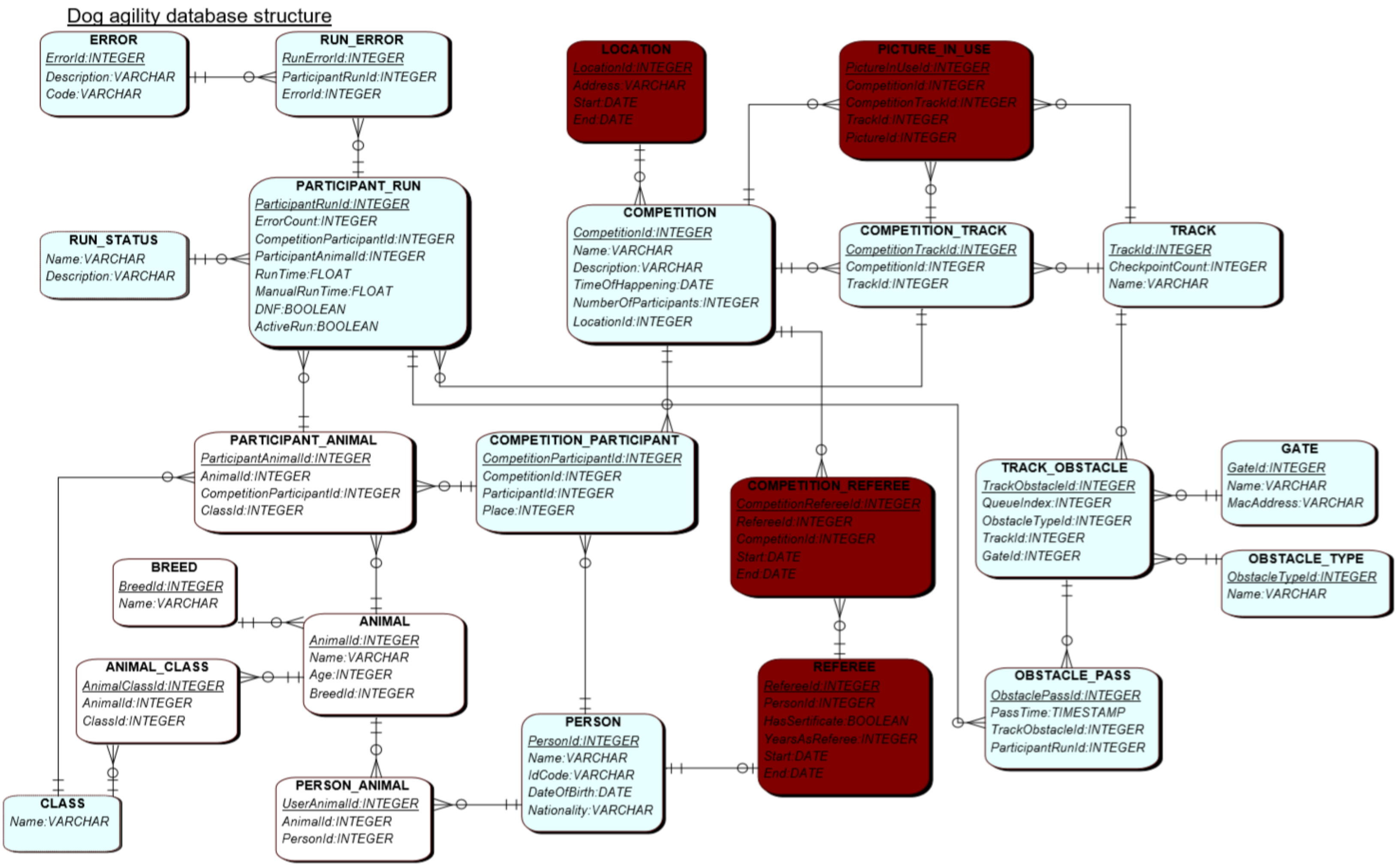
Also, the Unit of Work (UOW) is implemented. UOW is handling saving data to database. CRUD operations are done by repositories.

Summary

Used sources

1. https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design
2. https://deviq.com/repository-pattern/

ERD


Dog agility database structure
ERROR
ErrorId:INTEGER
Description:VARCHAR
Code:VARCHAR
RUN_ERROR
RunErrorId:INTEGER
ParticipantRunId:INTEGER
ErrorId:INTEGER
LOCATION
LocationId:INTEGER
Address:VARCHAR
Start:DATE
End:DATE
PICTURE_IN_USE
PictureInUseId:INTEGER
CompetitionId:INTEGER
CompetitionTrackId:INTEGER
TrackId:INTEGER
PictureId:INTEGER
PARTICIPANT_RUN
ParticipantRunId:INTEGER
ErrorCount:INTEGER
CompetitionParticipantId:INTEGER
ParticipantAnimalId:INTEGER
RunTime:FLOAT
ManualRunTime:FLOAT
DNF:BOOLEAN
ActiveRun:BOOLEAN
RUN_STATUS
Name:VARCHAR
Description:VARCHAR
COMPETITION
CompetitionId:INTEGER
Name:VARCHAR
Description:VARCHAR
TimeOfHappening:DATE
NumberOfParticipants:INTEGER
LocationId:INTEGER
COMPETITION_TRACK
CompetitionTrackId:INTEGER
CompetitionId:INTEGER
TrackId:INTEGER
TRACK
TrackId:INTEGER
CheckpointCount:INTEGER
Name:VARCHAR
PARTICIPANT_ANIMAL
ParticipantAnimalId:INTEGER
AnimalId:INTEGER
CompetitionParticipantId:INTEGER
ClassId:INTEGER
COMPETITION_PARTICIPANT
CompetitionParticipantId:INTEGER
CompetitionId:INTEGER
ParticipantId:INTEGER
Place:INTEGER
COMPETITION_REFEREE
CompetitionRefereeId:INTEGER
RefereeId:INTEGER
CompetitionId:INTEGER
Start:DATE
End:DATE
TRACK_OBSTACLE
TrackObstacleId:INTEGER
QueueIndex:INTEGER
ObstacleTypeId:INTEGER
TrackId:INTEGER
GateId:INTEGER
GATE
GateId:INTEGER
Name:VARCHAR
MacAddress:VARCHAR
OBSTACLE_TYPE
ObstacleTypeId:INTEGER
Name:VARCHAR
BREED
BreedId:INTEGER
Name:VARCHAR
ANIMAL
AnimalId:INTEGER
Name:VARCHAR
Age:INTEGER
BreedId:INTEGER
ANIMAL_CLASS
AnimalClassId:INTEGER
AnimalId:INTEGER
ClassId:INTEGER
REFEREE
RefereeId:INTEGER
PersonId:INTEGER
HasSertificate:BOOLEAN
YearsAsReferee:INTEGER
Start:DATE
End:DATE
OBSTACLE_PASS
ObstaclePassId:INTEGER
PassTime:TIMESTAMP
TrackObstacleId:INTEGER
ParticipantRunId:INTEGER
PERSON
PersonId:INTEGER
Name:VARCHAR
IdCode:VARCHAR
DateOfBirth:DATE
Nationality:VARCHAR
CLASS
Name:VARCHAR
PERSON_ANIMAL
UserAnimalId:INTEGER
AnimalId:INTEGER
PersonId:INTEGER