TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Kersti Miller 175726IDDR

PORTFOLIO MANAGER

Scope of work in Distributed Systems project

Supervisor: Andres Käver

Tallinn 2020

Author’s declaration of originality

I hereby certify that I am the sole author of this report. 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: Kersti Miller

06.03.2020

Abstract

This project proposal is written in english and is 32 pages long, including 5 chapters, 17 figures and 6 tables.

List of abbreviations and terms

PK	Primary key
FK	Foreign key
1:0-1	One-to-Zero or One-to-One relationship
1:M	One-to-Many relationship

Table of Contents

Author's declaration of originality....2
Abstract....3
List of abbreviations and terms....4
List of tables....7
1 Introduction....8
2 Project Scopes....9
2.1 Scope SMALL....9
2.2 Scope MEDIUM....12
2.3 Scope LARGE....14
3 ERD schema....17
4 Soft delete and soft update....18
4.1 Test tables....18
4.1.1 1....18
4.1.2 1:M relationship....18
4.1.3 1:1-0 relationship....19
4.2 Insert-Only Database....19
4.2.1 Test case: 1 table....19
4.2.2 Test case: 1:M relationship....20
4.2.3 Test case: 1: 0-1 relationship....23
4.2.4 Pros and cons....24
4.3 History tables....25
4.3.1 Test case: 1 table....25
4.3.2 Test case: 1:M relationship....26
4.3.3 Test case: 1:0-1 relationship....28
4.3.4 Pros and cons....29
4.4 Conclusions....30
5 Summary....31
References....32

List of Figures

Figure 1: Scope SMALL: initial dashboard view.....9
Figure 2: Scope SMALL: changing the data.....10
Figure 3: Scope SMALL: adding new asset with new type and platform.....10
Figure 4: Scope SMALL: adding new platform.....11
Figure 5: Scope SMALL: filled fields for adding a new asset.....11
Figure 6: Scope SMALL: new platform is displayed in the dashboard table.....12
Figure 7: Scope MEDIUM initial dashboard view.....13
Figure 8: Scope MEDIUM: changing the balance and adding the details.....13
Figure 9: Scope LARGE: dashboard view with data changing.....14
Figure 10: Scope LARGE: adding shared asset to existing asset.....15
Figure 11: Scope LARGE: viewing specific assets distributions and parts.....15
Figure 12: Scope LARGE: initial statistics list.....16
Figure 13: Scope LARGE: initial groups page.....16
Figure 14: ERD schema.....17
Figure 15: Location.....18
Figure 16: Location-Asset 1:M relationship.....18
Figure 17: Person-Photo 1:1-0 relationship.....19

List of tables

Table 1: Selected results and queries from 1 table soft delete and soft update.....20
Table 2: Selected results and queries from 1:M tables soft delete and soft update.....22
Table 3: Selected results and queries from 1:0-1 tables soft delete and soft update.....24
Table 4: Selected results and queries from 1 table soft delete and soft update.....26
Table 5: Selected results and queries from 1:M tables soft delete and soft update.....28
Table 6: Selected results and queries from 1:0-1 tables soft delete and soft update.....29

1 Introduction

The goal of this project is to create a portfolio manager for assets. The need for a proper assets manager came from authors personal need to track and follow her family portfolio. Author has tried several portfolio managers, but they have not filled all the needs and wishes upon the service provided. This project is aiming to move towards easier management from sophisticated excel sheets to the proper application.

The author has created three scopes for the project: small, medium and large. The scopes will be described in more detail in chapter 2. They represent the progress of the project and if all goes well enough there might be scope named extra-large. The scopes are subject to change during the process.

The small scope of this project is to create a solution where person can track it's current state of the asset values. User can see the total balance of all the assets according to the date the data was inserted. All the data will be inserted manually. That scope is rather small and it's created for to be the starting point for development.

The medium scope will have possibility to differentiate different parts of one asset. For example when user is buying stocks from company X three times a year, then each transaction will be visible.

Larger scope of this project is to have possibility to create groups since assets can be shared between family members and friends. Hopefully the author can reach to that scope.

In an ideal world the data should be fetched from the API-s of the service providers (different platforms where users can invest and trade). In reality there aren't many API-s available.

2 Project Scopes

2.1 Scope SMALL

The small scope of this project is to create a solution where person can track it's current state of the asset values. User can see the total balance of all the assets according to the date the data was inserted. All the data will be inserted manually. User can insert Goals and Tasks, but they are not tied with any certain asset. The table is showing the sum of all the assets and user can not distinguish sub assets. For example user has 15 000 EUR in Funderbeam in different projects, but they are not identifiable in small scope. The small scope does not have enough complexity to fulfill the project needs.

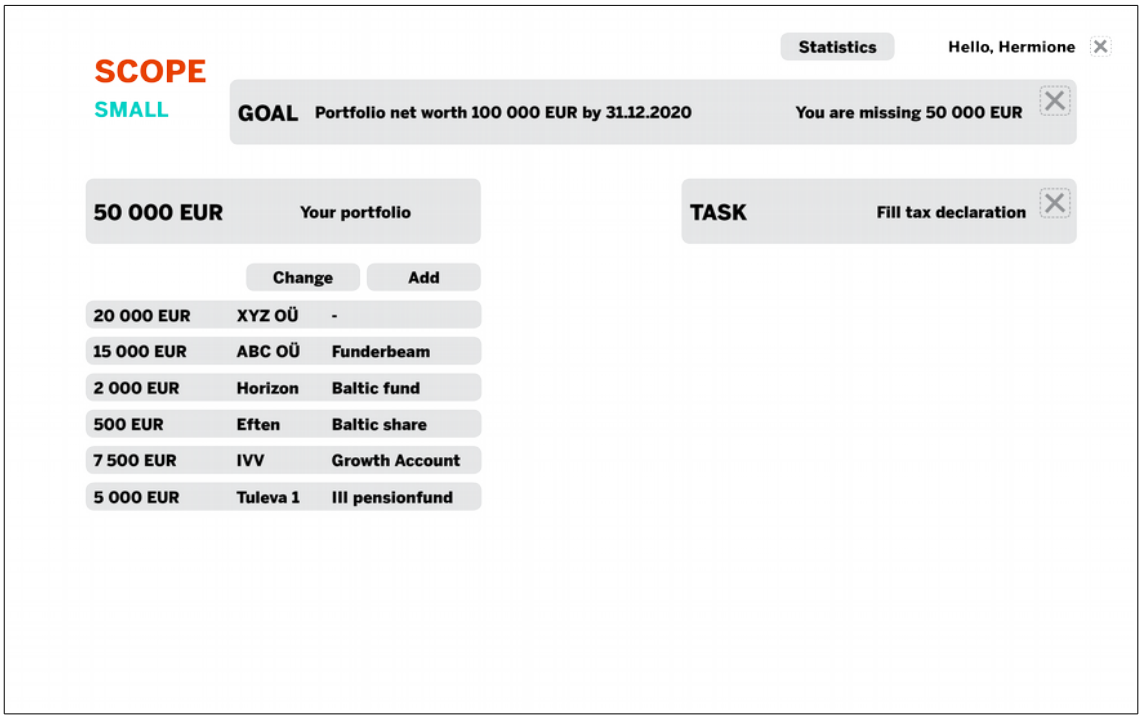

Figure 1: Scope SMALL: initial dashboard view

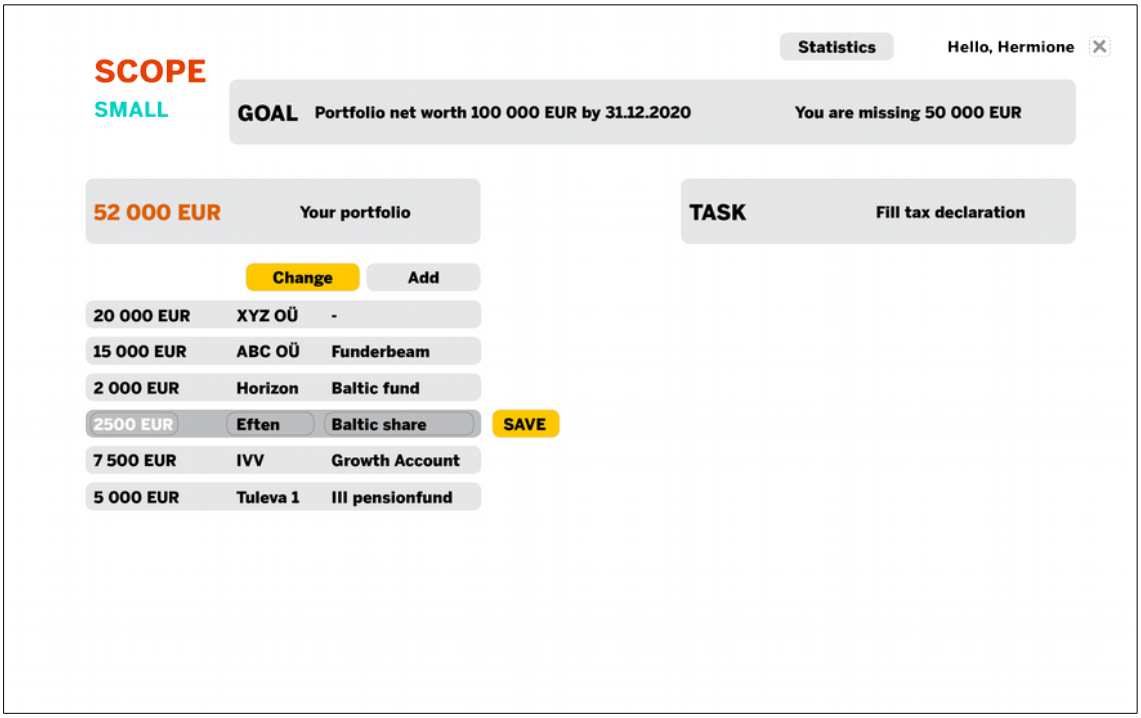

Figure 2: Scope SMALL: changing the data

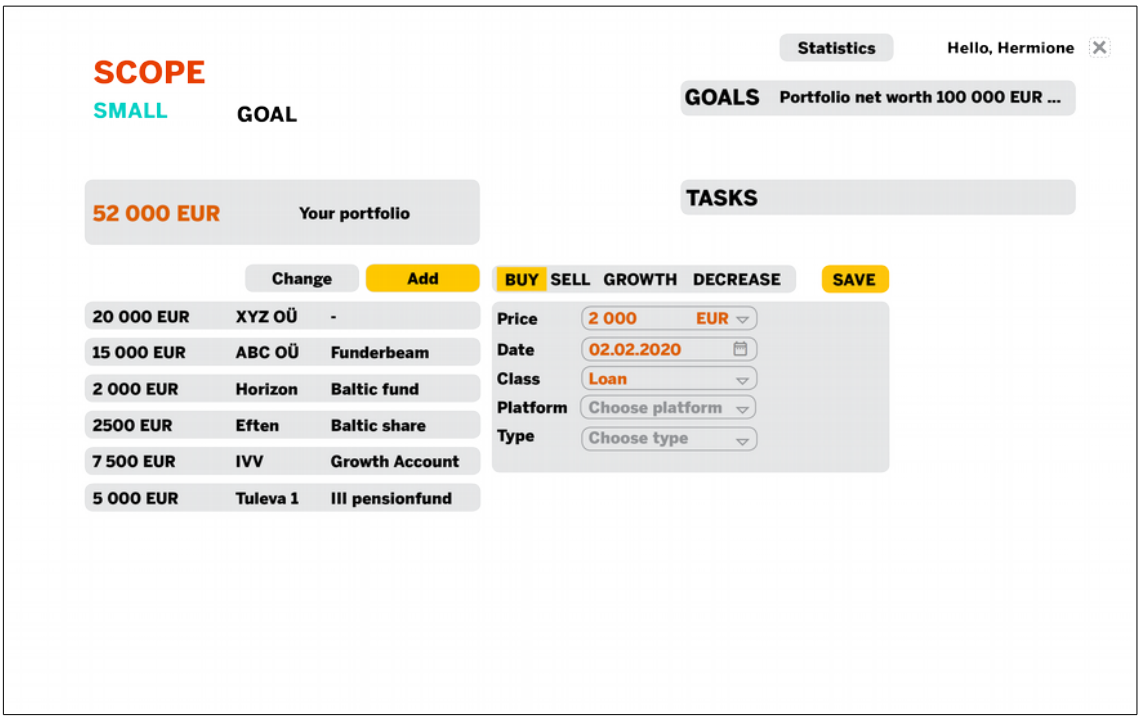

Figure 3: Scope SMALL: adding new asset with new type and platform

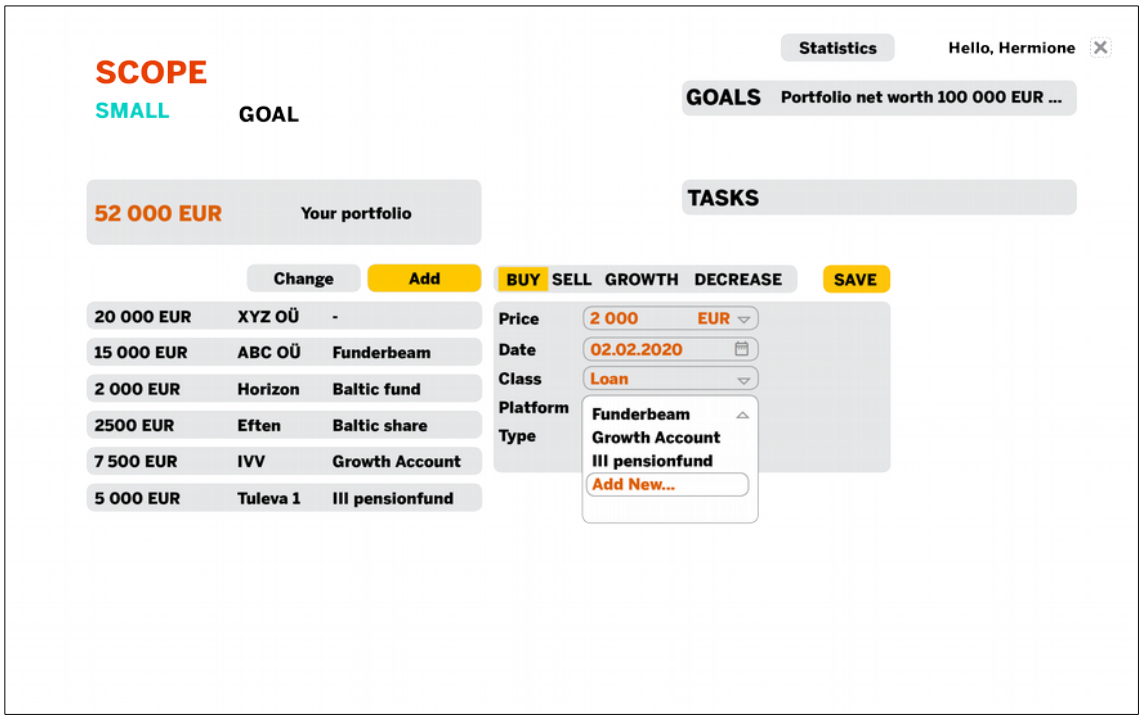

Figure 4: Scope SMALL: adding new platform

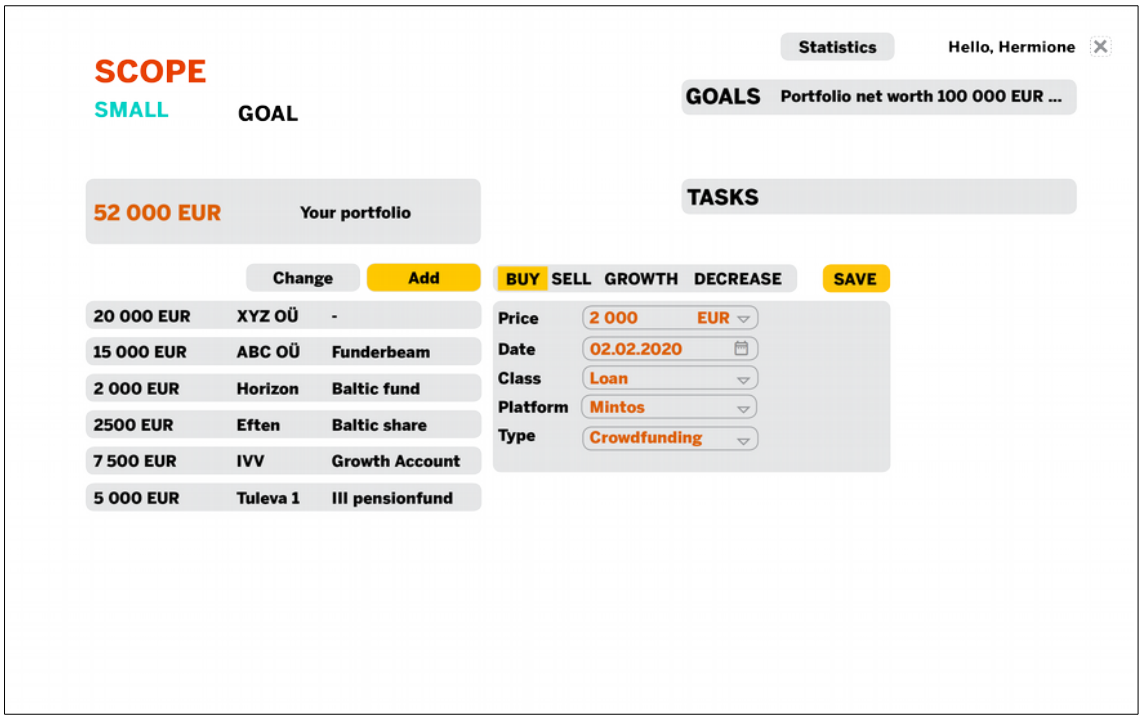

Figure 5: Scope SMALL: filled fields for adding a new asset

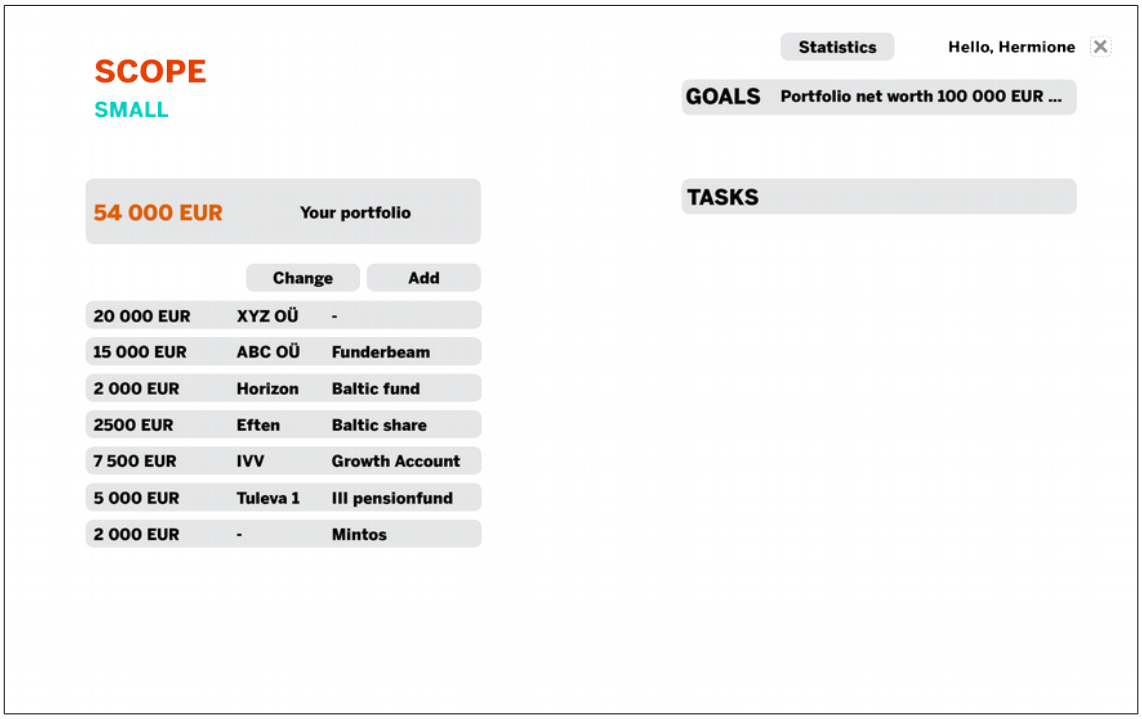

Figure 6: Scope SMALL: new platform is displayed in the dashboard table

2.2 Scope MEDIUM

The medium scope will be the minimum for that project. It will have the possibility to display different stats on the assets table. For example it could show how portfolio is distributed. How large is every asset share in the portfolio and what is the state compared to previous data insertion: has the position decreased, gained value or if the value is the same.

Medium scope will have transactions associated with assets and then user can insert if he had an income or outgoing, was the received money interests, dividends or did the user pays some service fees.

The tasks can be associated with assets or transactions. The goals can be related with assets class, location or location type. Goals card will display calculated numbers and recommendations for how to reach your goal and how much time do you have until due date.

There will be statistics page with more complex data.

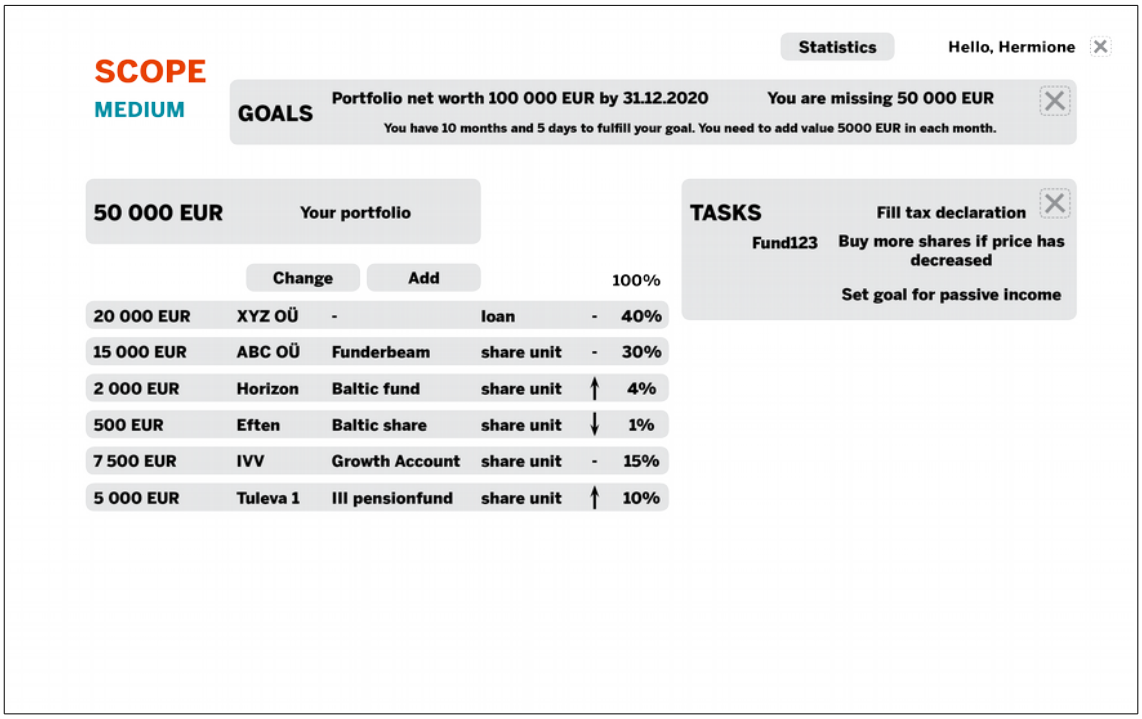

Figure 7: Scope MEDIUM initial dashboard view

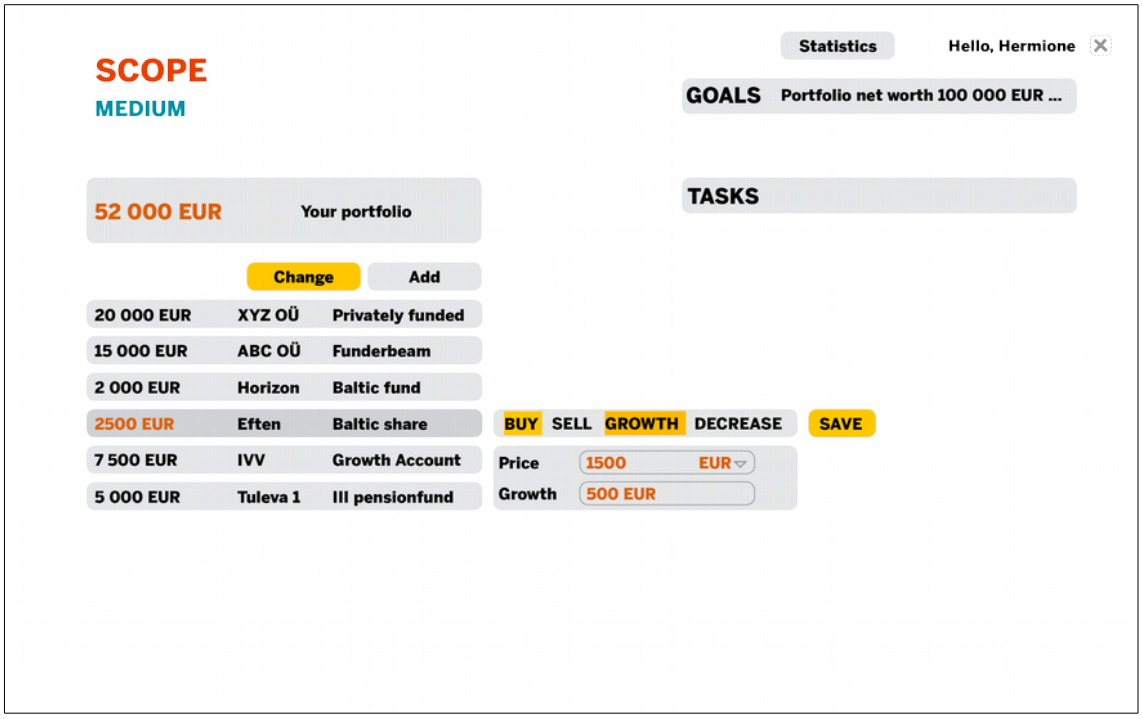

Figure 8: Scope MEDIUM: changing the balance and adding the details

2.3 Scope LARGE

The large scope of this project is to have possibility to create groups since assets can be shared between family members and friends. User can be a private person and it could also have a company where it can make it's investments.

User can make an entry about new assets and the program should calculate missing data by itself. For example if user inserts an entry where user has bought shares worth of 1500 euros and inserts the amount of shares, the price of one share will be calculated and vice versa about the other components.

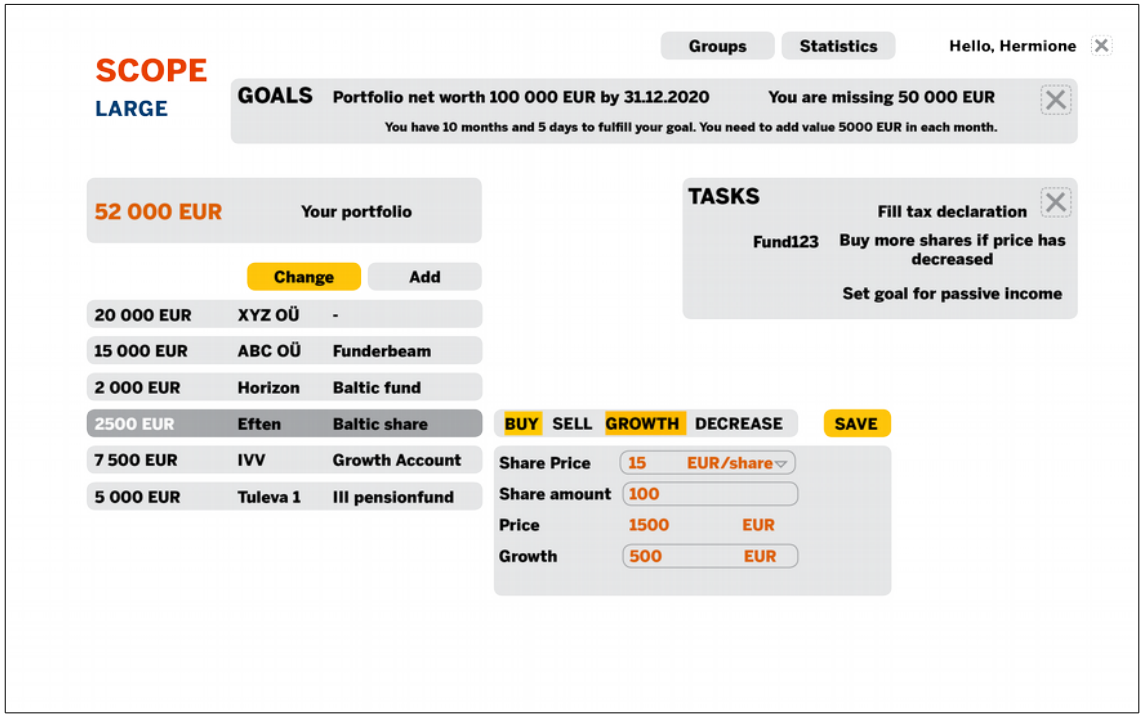

Figure 9: Scope LARGE: dashboard view with data changing

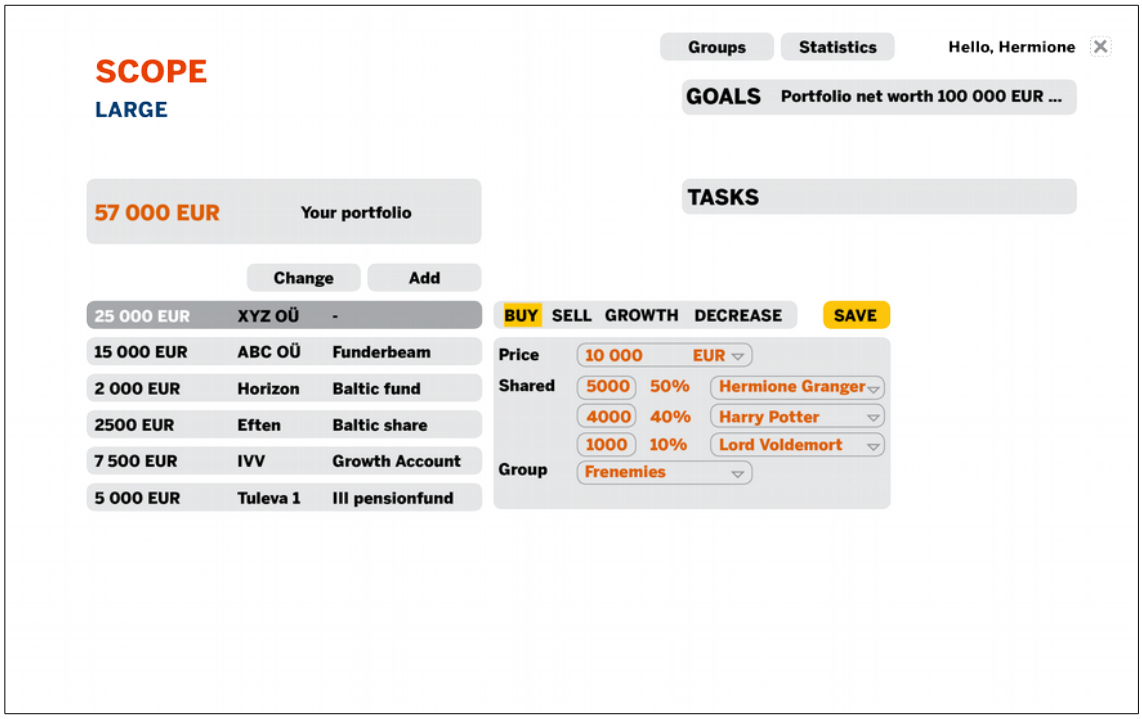

Figure 10: Scope LARGE: adding shared asset to existing asset.

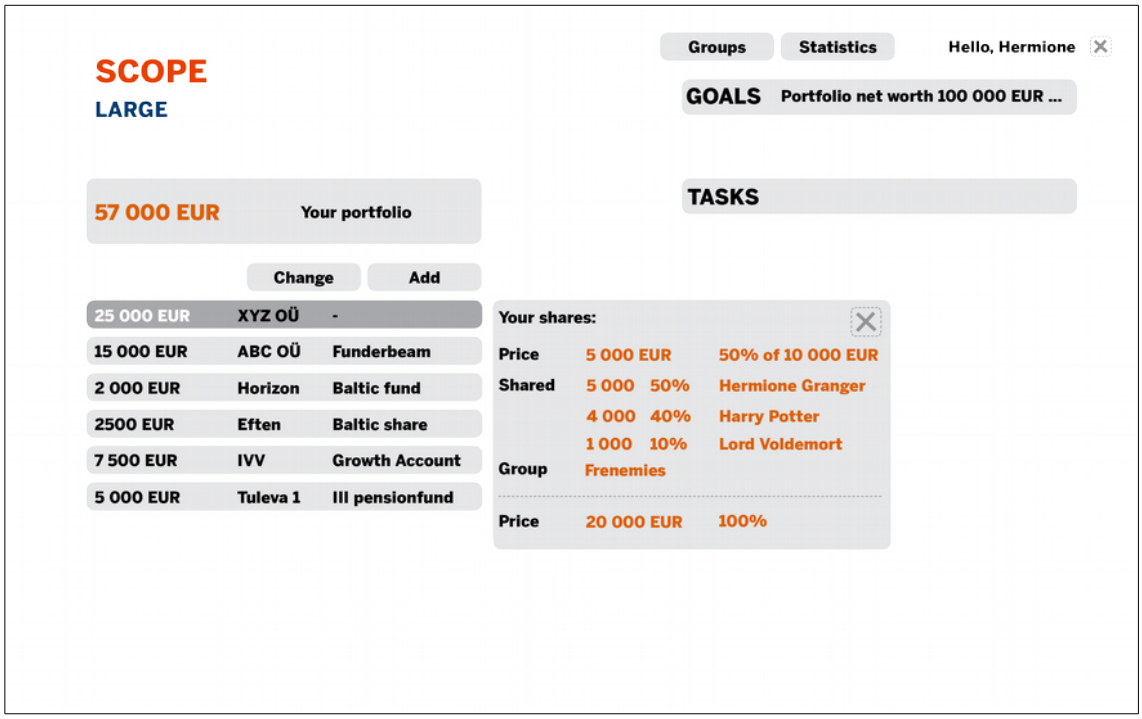

Figure 11: Scope LARGE: viewing specific assets distributions and parts

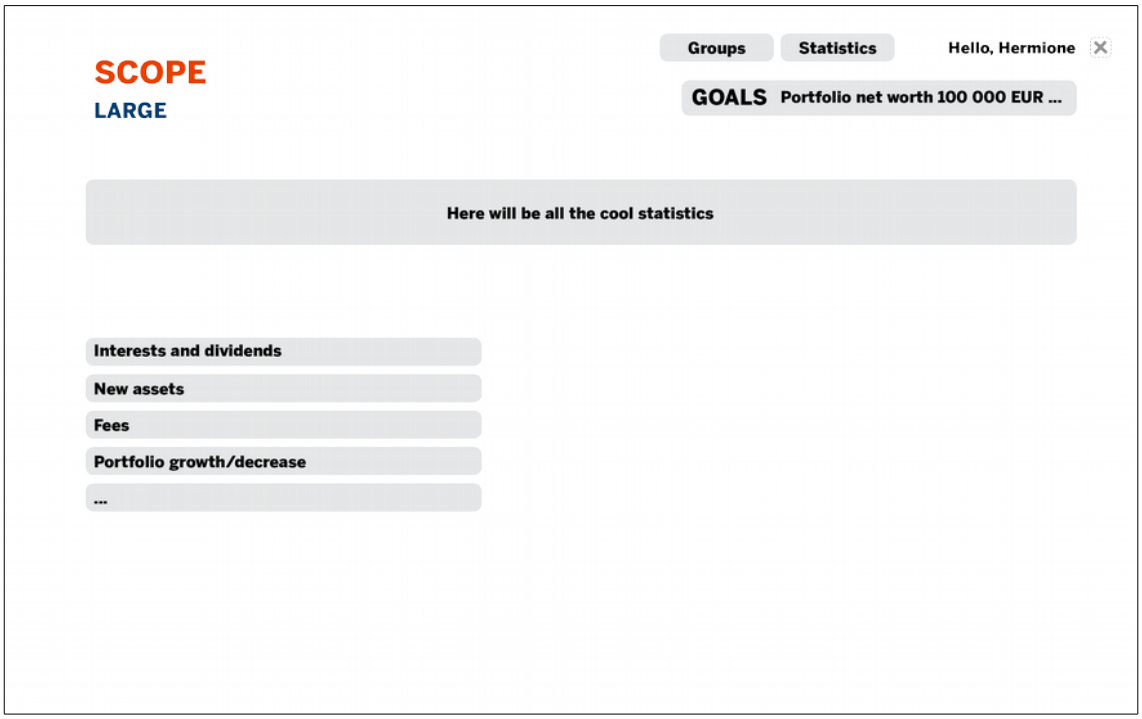

Figure 12: Scope LARGE: initial statistics list

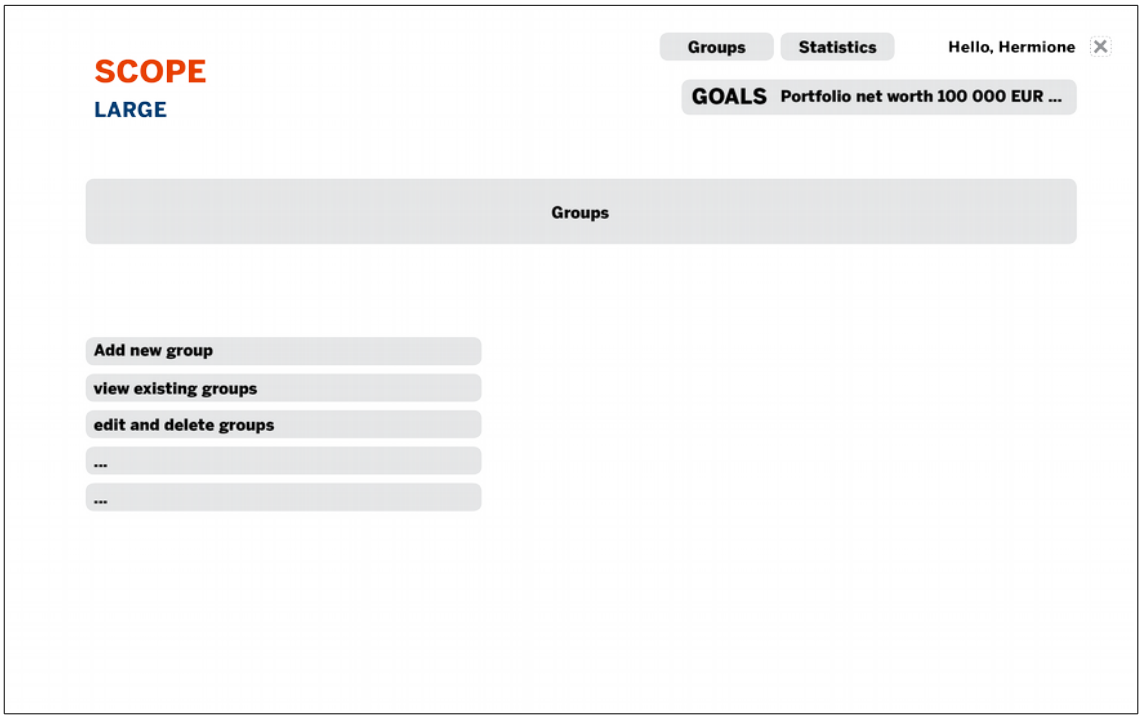

Figure 13: Scope LARGE: initial groups page

3 ERD schema

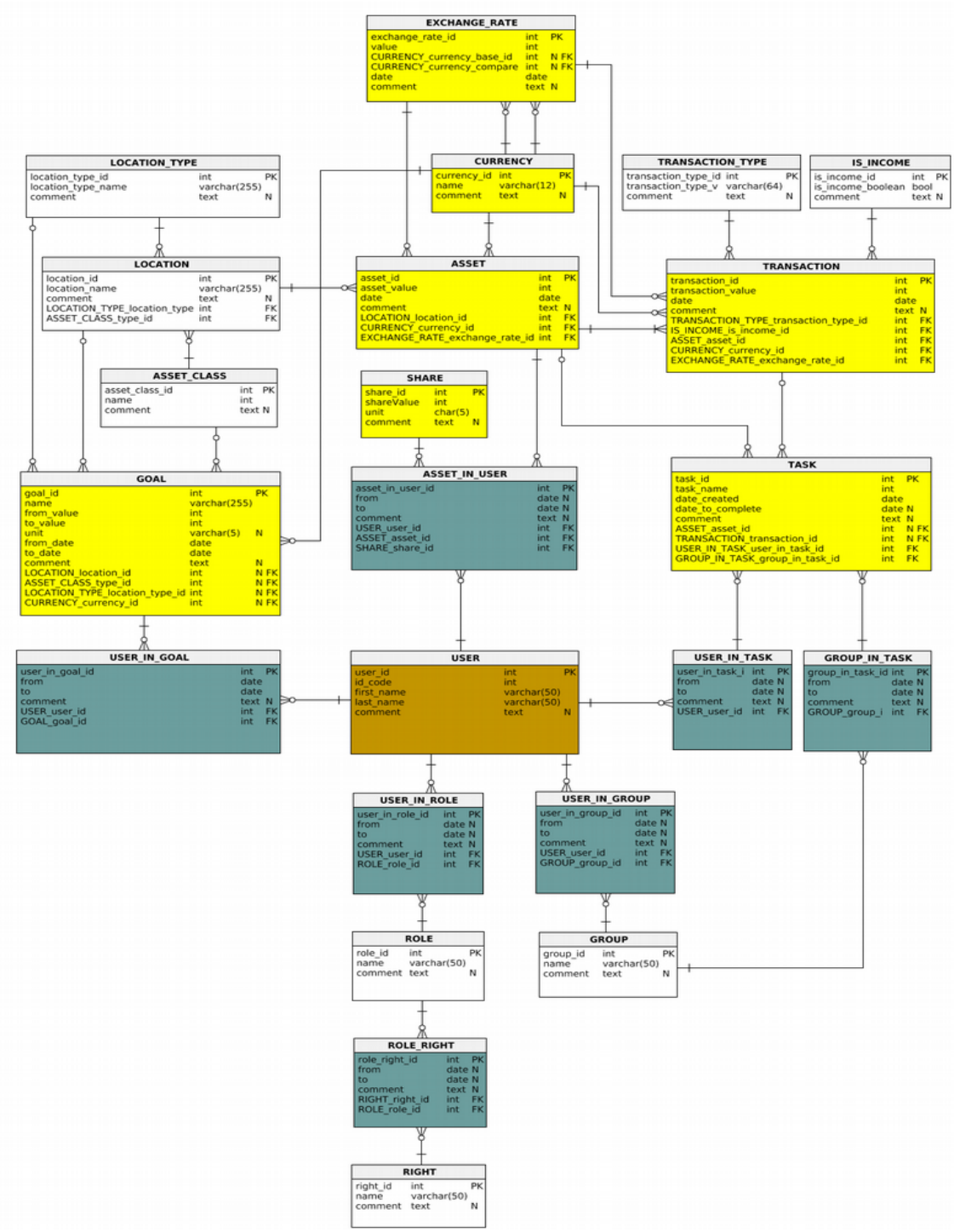

Figure 14: ERD schema

4 Soft delete and soft update

One of the main tasks for the project was to decide how to proceed with the revisionable database. This means the implementation of the techniques that are called among other names as soft delete and soft update. In authors project there will be discussion about two approaches.

1. Insert-Only Database
2. History tables

4.1 Test tables

For testing out previously mentioned approaches in database design, author has created three table sets.

4.1.1 1

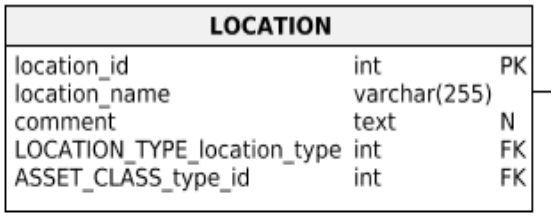

Figure 15: Location

4.1.2 1:M relationship

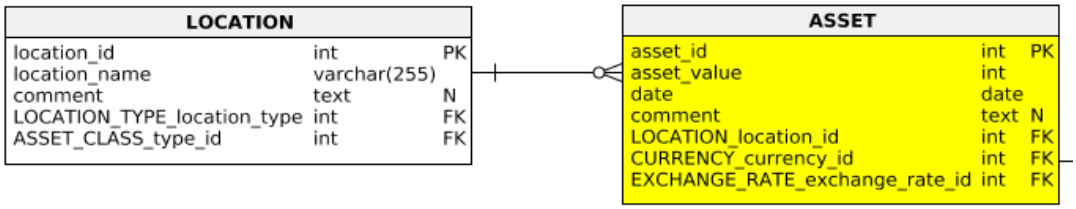

Figure 16: Location-Asset 1:M relationship

4.1.3 1:1-0 relationship

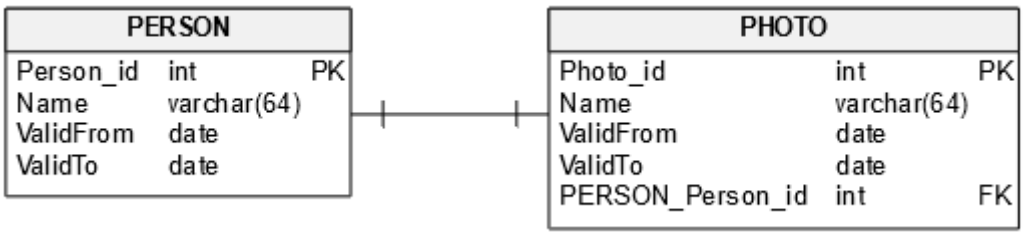

Figure 17: Person-Photo 1:1-0 relationship

4.2 Insert-Only Database

First approach for database design is the idea that you never update or delete data, only the validity column. Each table has two "timestamp" or "datetime" columns that together describe validity period: "valid-from" and "valid-to". They start with the predefined value for the beginning of time and the end of time. When user needs to „update" the row in any way, it updates the previous data "valid-to" value to correct date and then adds a new row with updated data which has now "valid-from" the same date as the previous data row "valid-to". It is recommended to have a unique index or composite key out of the foreign key(s) and the "to" value since then user can't insert new row before updating the previous row "valid-to" value. With query you should look for data that has "valid-to" predefined end of time value.[1]

4.2.1 Test case: 1 table

With only one table author does not need composite keys and data insertion is easier. PK is auto-incremented.

Test table:

```
CREATE TABLE LocationOnly (
     LocationOID INT             IDENTITY PRIMARY KEY,
     Name        VARCHAR(64)     NOT NULL,
     ValidFrom   DATE            NOT NULL,
     ValidTo     DATE            NOT NULL,
   )
```

[1] "Ideas on database design for capturing audit trails". – *StackOverflow,* 26. VI 2009, https://stackoverflow.com/questions/1051449/ideas-on-database-design-for-capturing-audit-trails, used 06. III 2020.

Steps to perform soft update:

1. INSERT new row with old name and add “ValidFrom” and “ValidTo” dates
2. UPDATE the original row name

In order to soft delete there is one step extra:

1. INSERT new row with old name and add “ValidFrom” and “ValidTo” dates
2. UPDATE the original row name next ValidTo date

	LocationOID	Name	ValidFrom	ValidTo
1	1	Eften	2020-03-01	9999-12-31
2	2	LHV	2020-01-01	9999-12-31
3	3	ABC	2020-02-01	2020-04-01
4	4	Eften	2020-01-01	2020-02-01
5	5	Eften III	2020-02-01	2020-03-01
6	6	ABC	2020-02-01	2020-03-01

```
-- CURRENT TIME
DECLARE @CurrentTime DATETIME2
SELECT @CurrentTime = '2020-05-01'

-- SELECT ALL
SELECT * FROM LocationOnly
```

	LocationOID	Name	ValidFrom	ValidTo
1	1	Eften	2020-03-01	9999-12-31
2	2	LHV	2020-01-01	9999-12-31

```
-- SELECT VALID
SELECT * FROM LocationOnly
  WHERE LocationOnly.ValidTo > @CurrentTime
```

	LocationOID	Name	ValidFrom	ValidTo
1	3	ABC	2020-02-01	2020-04-01
2	4	Eften	2020-01-01	2020-02-01
3	5	Eften III	2020-02-01	2020-03-01
4	6	ABC	2020-02-01	2020-03-01

```
-- SELECT INVALID
SELECT * FROM LocationOnly
  WHERE LocationOnly.ValidTo < @CurrentTime
```

Table 1: Selected results and queries from 1 table soft delete and soft update

4.2.2 Test case: 1:M relationship

Author has two tables which will have different data loads: Location and Asset. In a portfolio manager the more used table among mentioned two is Asset. It will have all the assets related to one location. Less used is Location. In one location user can have many assets. Table where data changes or additions will happen rather rarely does not need complicated system for tracking changes which in our case is Location. One must be more careful with finding suitable database design for Assets.

Test tables:

```
CREATE TABLE Location (
  LocationID  INT             NOT NULL,
  Name        VARCHAR (64)    NOT NULL,
  ValidFrom   DATE            NOT NULL,
  ValidTo     DATE            NOT NULL,
  CONSTRAINT  PK_Location     PRIMARY KEY (LocationID, ValidTo)
)

CREATE TABLE Asset (
  AssetId     INT             NOT NULL,
  Value       INT             NOT NULL,
  ValidFrom   DATE            NOT NULL,
  ValidTo     DATE            NOT NULL,
  LocationID  INT,
  LocationValidTo DATE,
  CONSTRAINT  PK_Asset             PRIMARY KEY (AssetID, ValidTo),
  CONSTRAINT  FK_Asset_Location    FOREIGN KEY (LocationID, LocationValidTo) REFERENCES Loca-
tion(LocationID, ValidTo)
)
```

Author has declared composite PK-s and FK-s from „ID" and „ValidTo" columns. With that approach one has to in case of soft update Location Name:

3. INSERT new row with old name and add "ValidFrom" and "ValidTo" dates.
4. UPDATE the original row name

Then there will be no conflicts between composite PK and FK in the Assets table. This has to be done like this in order to ensure that there is a match and only one match for Asset in Locations table.

In order to soft delete there is one step extra:

1. INSERT copy with ValidTo date
2. UPDATE Assets table records with new LocationValidTo
3. UPDATE Locations copy with another ValidTo date

Fraction of inserted data and some query result:

	Value	Name	ValidFrom	ValidTo
1	10000	Eften	2020-01-01	2020-02-01
2	10000	Eften III	2020-02-01	2020-03-01
3	10000	Eften	2020-03-01	3000-01-01
4	2000	LHV	2020-01-01	3000-01-01
5	5000	LHV	2020-01-01	3000-01-01
6	2500	Eften	2020-01-01	2020-02-01
7	2500	Eften III	2020-02-01	2020-03-01
8	2500	Eften	2020-03-01	3000-01-01
9	500	ABC	2020-02-01	2020-03-01
…	500	ABC	2020-02-01	2020-04-01

```
-- CURRENT TIME
DECLARE @CurrentTime DATETIME2
SELECT @CurrentTime = '2020-05-01'

-- SELECT ALL
SELECT
   Asset.Value,
   Location.Name,
   Location.ValidFrom,
   Location.ValidTo
FROM Asset
JOIN Location
ON Asset.LocationID = Location.LocationID
```

	Asset Value	Name	ValidTo
1	12500	Eften	3000-01-01
2	7000	LHV	3000-01-01

```
-- SELECT VALID AND SUMMED
SELECT
   SUM(Asset.Value) AS 'Asset Value',
   Location.Name,
   Location.ValidTo
FROM Asset
INNER JOIN Location
ON Asset.LocationID = Location.LocationID
AND Asset.LocationValidTo = Location.ValidTo
WHERE Location.ValidTo > @CurrentTime
GROUP BY Location.Name, Location.ValidTo
```

	Value	Name	ValidFrom	ValidTo
1	10000	Eften	2020-01-01	2020-02-01
2	2500	Eften	2020-01-01	2020-02-01
3	10000	Eften III	2020-02-01	2020-03-01
4	2500	Eften III	2020-02-01	2020-03-01
5	500	ABC	2020-02-01	2020-03-01
6	500	ABC	2020-02-01	2020-04-01

```
-- SELECT INVALID
SELECT
   Asset.Value,
   Location.Name,
   Location.ValidFrom,
   Location.ValidTo
FROM Asset
JOIN Location
ON Asset.LocationID = Location.LocationID
WHERE Location.ValidTo < @CurrentTime
```

Table 2: Selected results and queries from 1:M tables soft delete and soft update

4.2.3 Test case: 1: 0-1 relationship

Author created fictive test case from Person-Photo relationship. One person can have none or one photos. Every photo has a person.

Test tables:

```
CREATE TABLE Person (
    PersonID    INT             NOT NULL,
    Name        VARCHAR(64)     NOT NULL,
    ValidFrom   DATE            NOT NULL,
    ValidTo     DATE            NOT NULL,
    CONSTRAINT  PK_Person       PRIMARY KEY (PersonID, ValidTo)
  )

  CREATE TABLE Photo (
    PhotoId         INT             NOT NULL,
    Name            VARCHAR(64)     NOT NULL,
    ValidFrom       DATE            NOT NULL,
    ValidTo         DATE            NOT NULL,
    PersonID        INT,
    PersonValidTo   DATE,
    CONSTRAINT      PK_Photo        PRIMARY KEY (PhotoId, ValidTo),
    CONSTRAINT      FK_Photo_Person UNIQUE (PersonID, PersonValidTo)
  )
```

The steps to perform soft delete and soft update are the same as in 1:M relationship.

Soft update:

1. INSERT new row with old name and add “ValidFrom” and “ValidTo” dates.
2. UPDATE the original row name

Soft delete:

1. INSERT copy with ValidTo date
2. UPDATE Photo table records with new PersonValidTo
3. UPDATE Person copy with another ValidTo date

Inserted data and some query result:

	Name	Name	ValidFrom	ValidTo
1	Peeter	Passipilt	2020-01-01	9999-12-31
2	Norman	ID-pilt	2020-01-01	2020-02-01
3	Norman	ID-pilt	2020-01-01	2020-04-01
4	Norman	*NULL*	2020-01-01	2020-03-01
5	Meeli	*NULL*	2020-01-01	9999-12-31

```
-- CURRENT TIME
DECLARE @CurrentTime DATETIME2
SELECT @CurrentTime = '2020-05-01'

-- SELECT ALL
SELECT
   Person.Name,
   Photo.Name,
   Person.ValidFrom,
   Person.ValidTo
FROM Person
LEFT JOIN Photo
ON Photo.PersonID = Person.PersonID
ORDER BY Photo.ValidTo DESC
```

	Name	Name	ValidFrom	ValidTo
1	Passipilt	Peeter	2020-01-01	9999-12-31

```
-- SELECT VALID
SELECT
   Photo.Name,
   Person.Name,
   Person.ValidFrom,
   Person.ValidTo
FROM Photo
JOIN Person
 ON Photo.PersonID = Person.PersonID
AND Photo.PersonValidTo = Person.ValidTo
WHERE Person.ValidTo > @CurrentTime
```

	Name	Name	ValidFrom	ValidTo
1	ID-pilt	Norman	2020-01-01	2020-02-01
2	ID-pilt	Norman	2020-01-01	2020-04-01

```
-- SELECT INVALID
SELECT
   Photo.Name,
   Person.Name,
   Person.ValidFrom,
   Person.ValidTo
FROM Photo
JOIN Person
 ON Photo.PersonID = Person.PersonID
 WHERE Person.ValidTo < @CurrentTime
```

Table 3: Selected results and queries from 1:0-1 tables soft delete and soft update

4.2.4 Pros and cons

Pros:

- No separate tables
- Only one row of some certain data is valid at the time and is visible via “valid-to” columns
- Does not have extra columns to track changes

Cons:

- A lot of data in one table
- Juggling between records for deleting

4.3 History tables

A history table is created for the purpose of using one table to track changes in another table. The idea is very simple, but when the concept is implemented in an inexperienced way it will lead to a data bloat and it would be difficult to make queries. That means that all the columns are being copied into the history table and they will include redundant repetitions which are hard to inspect and query. Another approach is to add into the history table only the columns that you know what you will definitely need. This raises an issue whether we really know for certainty what columns are needed also in the future.[1]

Kenneth Downs has pointed out in his blog that the best performing and most secure method is to implement history tables with triggers on the source table as it is the best way to implement both security and the actual business rules in one encapsulated object (the table).[2] Author of this project does not have any knowledge about triggers and designing the database in the code hence the author can't make any decisions upon previous experience.

4.3.1 Test case: 1 table

With only one table author does not need composite keys and data insertion is easier. PK is auto-incremented.

Test tables:

```
CREATE TABLE LocationOnlyClean (

     LocationCID INT             IDENTITY PRIMARY KEY,
     Name        VARCHAR(64)     NOT NULL,
   )
CREATE TABLE LocationOnlyHistory (
     LocationHID INT             IDENTITY PRIMARY KEY,
     LocationCID INT             NOT NULL,
     Name_old    VARCHAR(64)     NOT NULL,
     Name_new    VARCHAR(64)     NOT NULL,
     Activity    VARCHAR(64)     NOT NULL,
     ActivityDate DATE           NOT NULL,
   )
```

[1] **Downs**, Kenneth. "History Tables". – *The Database Programmer*, 20. VII 2008, http://database-programmer.blogspot.com/2008/07/history-tables.html, used 06. III 2020.

[2] *Ibid*

Steps to perform soft update:

1. UPDATE the original row data
2. INSERT data about the actions in the history table

Soft update:

1. DELETE the original row data
2. INSERT data about the actions in the history table

	LocationCID	Name
1	2	LHV

```
-- CURRENT TIME
DECLARE @CurrentTime DATETIME2
SELECT @CurrentTime = '2020-05-01'

-- SELECT ALL / VALID
SELECT * FROM LocationOnlyClean
```

	LocationHID	LocationCID	Name_old	Name_new	Activity	ActivityDate
1	1	1	-	Eften	INSERT	2020-01-01
2	2	2	-	LHV	INSERT	2020-01-01
3	3	1	Eften	Eften III	UPDATE	2020-02-01
4	4	1	Eften III	-	DELETE	2020-03-01

```
-- SELECT HISTORY
SELECT * FROM LocationOnlyHistory
```

Table 4: Selected results and queries from 1 table soft delete and soft update

4.3.2 Test case: 1:M relationship

1:M test case is more complex, since every table needs a history table and more connections make hard to track data and actions.

Test tables:

```
CREATE TABLE LocationClean (
      LocationCID INT             IDENTITY PRIMARY KEY,
      Name        VARCHAR(64)     NOT NULL,
    )

CREATE TABLE AssetClean (
      AssetCId    INT             IDENTITY PRIMARY KEY,
      Value       INT             NOT NULL,
      LocationCID INT             NOT NULL,
      CONSTRAINT  FK_AssetC_LocationC  FOREIGN KEY (LocationCID) REFERENCES LocationClean(Loca-
tionCID)
    )

CREATE TABLE LocationHistory (
      LocationHID INT             IDENTITY PRIMARY KEY,
      LocationCID INT             NOT NULL,
      Name_old    VARCHAR(64)     NOT NULL,
      Name_new    VARCHAR(64)     NOT NULL,
      Activity    VARCHAR(64)     NOT NULL,
      ActivityDate DATE           NOT NULL,
    )
```

```
CREATE TABLE AssetHistory (
      AssetHId     INT              IDENTITY PRIMARY KEY,
      AssetCId     INT              NOT NULL,
      Value_old    INT              NOT NULL,
      Value_new    INT              NOT NULL,
      Value_diff   INT              NOT NULL,
      Activity     VARCHAR(64)      NOT NULL,
      ActivityDate DATE             NOT NULL,
      LocationCID INT               NOT NULL,
      LocationHID INT               NULL,
      CONSTRAINT   FK_AssetH_LocationH   FOREIGN KEY (LocationHID) REFERENCES LocationHistory(Loc-
ationHID)
    )
```

Steps to perform soft update:

1. UPDATE the original row data
2. INSERT data about the actions in the history table

Soft delete on location:

1. DELETE the original data
2. INSERT data about the actions in the history table
3. UPDATE AssetHistory LocationHistoryID to delete event in LocationHistory

All inserted data and some query result:

	LocationCID	Name
1	2	LHV

```
-- CURRENT TIME
DECLARE @CurrentTime DATETIME2
SELECT @CurrentTime = '2020-05-01'

-- SELECT ALL/VALID
 SELECT AssetClean.Value AS 'ASSET VALUE',
        LocationClean.Name AS 'LOCATION NAME'
 FROM AssetClean
 JOIN LocationClean
 ON AssetClean.LocationCID = LocationClean.LocationCID
```

	Asset Value	Asset name	Date inserted
1	18000	Eften	2020-01-01
2	2000	LHV	2020-01-01

```
-- SELECT INSERTED AND SUMMED
 SELECT
   SUM(AssetHistory.Value_new) AS 'Asset Value',
      LocationHistory.Name_new AS 'Asset name',
      LocationHistory.ActivityDate AS 'Date inserted'
   FROM AssetHistory
   INNER JOIN LocationHistory
   ON AssetHistory.LocationHID = LocationHistory.Loca-
tionHID
   WHERE NOT LocationHistory.Name_new = '-'
   GROUP BY LocationHistory.Name_new, LocationHis-
tory.ActivityDate
```

	Asset Value	Asset name	Date inserted
1	10000, 8000	Eften	2020-01-01

```
-- SELECT ASSET WITH LOCATION - INVALID
    SELECT
      STRING_AGG(AssetHistory.Value_new, ', ') AS
'Asset Value',
  LocationHistory.Name_new AS 'Asset name',
  LocationHistory.ActivityDate AS 'Date inserted'
  FROM AssetHistory
  INNER JOIN LocationHistory
   ON AssetHistory.LocationHID = LocationHistory.Loca-
tionHID
  WHERE NOT LocationHistory.Name_new = '-'
  AND NOT EXISTS (
    SELECT LocationCID
    FROM LocationClean
    WHERE LocationHistory.LocationHID = Location-
Clean.LocationCID)
 GROUP BY LocationHistory.Name_new, LocationHis-
tory.ActivityDate
```

Table 5: Selected results and queries from 1:M tables soft delete and soft update

4.3.3 Test case: 1:0-1 relationship

1:0-1 test case is similar to 1:M .

Test tables:

```
CREATE TABLE PersonClean (
       PersonCID    INT             IDENTITY PRIMARY KEY,
       Name         VARCHAR(64)     NOT NULL,
)

CREATE TABLE PersonHistory (
       PersonHID    INT             IDENTITY PRIMARY KEY,
       PersonCID    INT             NOT NULL,
       Name_old     VARCHAR(64)     NOT NULL,
       Name_new     VARCHAR(64)     NOT NULL,
       Activity     VARCHAR(64)     NOT NULL,
       ActivityDate DATE            NOT NULL,
)

CREATE TABLE PhotoClean (
       PhotoCID     INT             IDENTITY PRIMARY KEY,
       Name         VARCHAR(64)     NOT NULL,
       PersonCID    INT             NOT NULL,
       CONSTRAINT   FK_PhotoC_PersonC   UNIQUE (PersonCID)
)

CREATE TABLE PhotoHistory (
       PhotoHId     INT             IDENTITY PRIMARY KEY,
       PhotoCId     INT             NOT NULL,
       Name_old     VARCHAR(64)     NOT NULL,
       Name_new    VARCHAR(64)     NOT NULL,
       Activity     VARCHAR(64)     NOT NULL,
       ActivityDate DATE            NOT NULL,
       PersonCID    INT             NOT NULL,
       PersonHID    INT             NULL,
       CONSTRAINT   FK_PhotoH_PersonH UNIQUE (PersonHID)
)
```

Steps to perform soft update on updating the person or photo:

1. UPDATE the original row data
2. INSERT data about the actions in the history table

Soft delete on person:

1. DELETE the original person data
2. INSERT data about the actions in the history table
3. DELETE photo associated with the person
4. INSERT data about the actions in the history table

All inserted data and some query result:

	Name	Name
1	Marta	ID-pilt

```
-- CURRENT TIME
DECLARE @CurrentTime DATETIME2
SELECT @CurrentTime = '2020-05-01'

-- SELECT VALID
SELECT PersonClean.Name, PhotoClean.Name
    FROM PersonClean
    JOIN PhotoClean
    ON PhotoClean.PersonCID = PersonClean.PersonCID
```

	Name_new	Name_new
1	Passipilt	Marek
2	Passipilt	Jarek
3	ID-pilt	Marta
4	Niisama pilt	Marek
5	Niisama pilt	Jarek

```
-- SELECT ALL
SELECT PhotoHistory.Name_new, PersonHistory.Name_new
    FROM PhotoHistory
    JOIN PersonHistory
    ON PhotoHistory.PersonCID = PersonHistory.PersonCID
    WHERE NOT PersonHistory.Name_new = '-'
      AND NOT PhotoHistory.Name_new = '-'
```

Table 6: Selected results and queries from 1:0-1 tables soft delete and soft update

4.3.4 Pros and cons

Pros:

- Clean data in one table and history in another

Cons:

- Data duplication
- Complex querying
- Easy to make bloat

4.4 Conclusions

With only one table its very easy to use Insert-only table. With 1:M relationship or 1:0-1 it's also not too complicated and data inserting feels more intuitive than with history tables. Insert only database design seems more intuitive and easier to implement and maintain. History table has really simple idea, but it's complicated to implement and keep track of the tables and indexes. Writing queries were troublesome. In theory triggers should be used.

5 Project design pattern – Repository

The Repository pattern is an abstraction layer between business logic and data source layers. Its main purpose is to reduce complexity and make the rest of the code persistent ignorant.[1] A Repository queries the data source for the data, maps the data from the data source to a business entity, and persists changes in the business entity to the data source.[2] As it's an abstraction, it should always return whatever the layer above wants to work with.[3] Repository should not leak persistence specific information up to the caller (typically by exposed IQueryable<T>).[4]

When there are a large number of domain classes or heavy querying then adding this layer helps minimize duplicate query logic and write less error-prone code.[5] When using repositories, you are forced to use strongly typed business entities which leads again more convenient codebase.

With separation of concerns it allows you to write easier unit tests for business logic. You can mock the repository and queries and never handle the real data. Repository pattern reduces the complexity in your tests and allow you to specialize your tests for the current layer.[6]

The advantage of using a Repository pattern is that your backend database can be changed later to use a different technology without having to change the repository interface.

[1] "Repository Pattern, Done Right". Code Project. https://www.codeproject.com/articles/526874/repository-pattern-done-right, used 24. III 2020.
[2] https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff649690(v=pandp.10)?redirectedfrom=MSDN, used 24. III 2020.
[3] "Repository Pattern, Done Right". Code Project.
[4] *Ibid*.
[5] https://martinfowler.com/eaaCatalog/repository.html, used 24. III 2020.
[6] https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff649690(v=pandp.10)?redirectedfrom=MSDN, used 24. III 2020.

Pattern becomes useful only when you have a complex domain or large and complex enterprise data scenarios, otherwise the maintaining would be an overhead because of the great deal of isolation and encapsulation within the domain model.[7]

5.1 DAO - Data Access Object

DAO and Repository pattern are ways of implementing Data Access Layer (DAL). DAO is also an abstraction object between the business logic and data source layers: it allows you to access data, and same as Repository, it abstracts the database connection and communication and returns domain. It has the same advantage to switch the database without other layers knowing about it.

The main difference between the Repository and the DAO is that the DAO is at a lower level of abstraction and doesn't speak the common language of the domain.[2] There are also some differences in the way they are: DAO being a bit more flexible/generic, while Repository is a bit more specific and restrictive to a type only. A Repository can be used with DAO's, but DAO can never be used with Repository.[3]

Author uses Repository pattern since it's more specific and does not get bloated so easily with implementations that does not belong there.[4] Work in progress...

[7] Microsoft documentation. "Microsoft Application Architecture Guide, 2nd Edition". https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658117(v=pandp.10)?redirectedfrom=MSDN#Domain-ModelStyle, used 24. III 2020.
[2] **Cargnelutti**, Federico. "Domain-Driven Design: The Repository". – *Federico Cargnelutti*, 15. III 2009, https://blog.fedecarg.com/2009/03/15/domain-driven-design-the-repository/, used 24. III 2020.
[3] Stackoverflow. https://stackoverflow.com/questions/8550124/what-is-the-difference-between-dao-and-repository-patterns, used 24. III 2020.
[4] "Don't use DAO, use Repository. – *The Thinking in Objects*, 26. VIII 2012, "https://thinkinginobjects.com/2012/08/26/dont-use-dao-use-repository/, used 24. III 2020.

6 Summary

References

Cargnelutti, Federico. “Domain-Driven Design: The Repository”. – *Federico Cargnelutti*, 15. III 2009, https://blog.fedecarg.com/2009/03/15/domain-driven-design-the-repository/, used 24. III 2020.

Downs, Kenneth. “History Tables”. – *The Database Programmer*, 20. VII 2008, http://database-programmer.blogspot.com/2008/07/history-tables.html, used 06. III 2020.

“Ideas on database design for capturing audit trails”. – *StackOverflow,* 26. VI 2009, https://stackoverflow.com/questions/1051449/ideas-on-database-design-for-capturing-audit-trails, used 06. III 2020.

“Don’t use DAO, use Repository. – *The Thinking in Objects*, 26. VIII 2012,

”https://thinkinginobjects.com/2012/08/26/dont-use-dao-use-repository/, used 24. III 2020.

Microsoft documentation. “Microsoft Application Architecture Guide, 2nd Edition”. https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658117(v=pandp.10)?redirectedfrom=MSDN#DomainModelStyle, used 24. III 2020

Stackoverflow. https://stackoverflow.com/questions/8550124/what-is-the-difference-between-dao-and-repository-patterns, used 24. III 2020.

“Repository Pattern, Done Right”. Code Project. https://www.codeproject.com/articles/526874/repository-pattern-done-right, used 24. III 2020.

https://martinfowler.com/eaaCatalog/repository.html, used 24. III 2020.