



TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY

Infotehnoloogia teaduskond

Tristan Krass, 179302IADB

Infort Information system

ICD0009, ICD0006 ja ICD0015

Project Description

Supervisor: Andres Käver

Contents

Introduction.....	4
--------------------------	----------

Author's declaration

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

Author: Tristan Krass

[pp.kk.aaaa]

Introduction

The goal of this project is to help speech therapist clinic to be more effective and make speech therapists life easier by enabling them to use Infront info system that enables them to add new information about their patients, manage registration and ability to use Estonian E-health services for various tasks.

The users will be doctors, therapists and clinic administrators. The users have the following functionalities - to accept cash and card payments or payment by transfer that is paid by medical institutions or the state, register for appointments, cancel appointments, choose the therapist, choose the appointment type, see the reports written by therapist by either visiting our platform or visiting the *digilugu*.

The first potential client to use this solution is *Kõneravi OÜ* who is currently using a 20 year old system called „Liisa“ and which is geared mostly towards hospitals and therefore is not an ideal solution for smaller clinics that only need few functionalities that are provided by this system. Also, there are some unmet needs like the ability to save, edit and send the generated receipts automatically to different medical institutions, hospitals and clients.

If the pilot project with *Kõneravi OÜ* is successful, then the goal is to find other small clinics that would need a similar solution. Therefore, it is important to make the system modular so it will be easy to add or remove new functionality given the need of the clinic.

The database that will be used for this project is Microsoft SQL Server. Rather than maintaining the database by myself I will be using a fully managed database service provided by Azure.

The backend services will be built on top of the database that will be the layer between the client app and the database layer. This is achieved by using the .Net Core latest framework. This approach – building a distributed backend service will enable to build various client facing application on top of this service.

Currently the plan is to build the client app as a Web application with React framework because of its big community and popularity. Some views and parts like admin interfaces are build by using the Razor page technology provided by Microsoft. Since most of the things that are done in admin interface are not dynamically changing, unlike the registration view. Also, it is not excluded that we will instead build Windows app since most of the clinics use Windows as their main operating system and some functionalities might need the underlying Windows API-s. The final decision about the client facing app will be decided in the future.

Initial Project ERD model

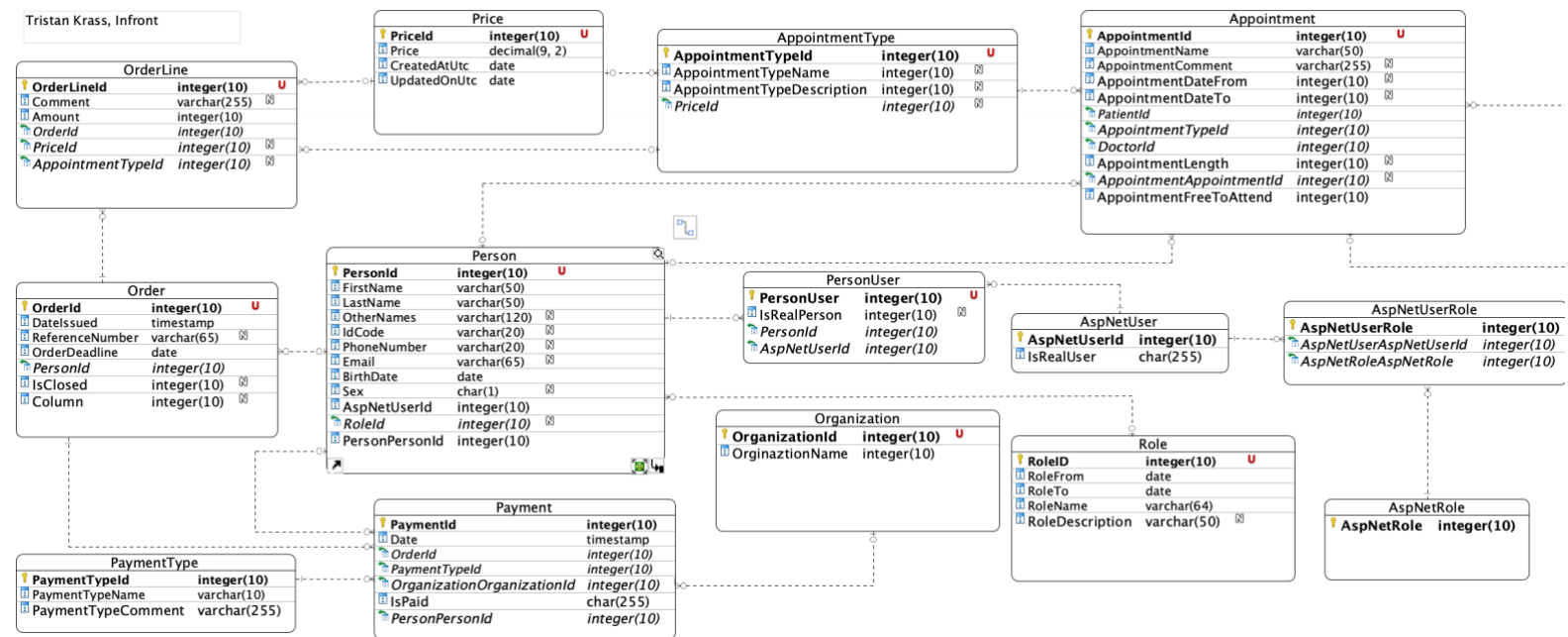


Figure 1. Initial Project diagram (using crow foot notation).

Analysis and testing of soft delete and update in MSSQL

First of all what is soft delete and soft update? As the “soft” implies it has to do something how the delete operation is executed. The simplest explanation is , with soft delete the data inside database will never get deleted and will never the data will never be updated only the copies of data will be changed. This means that if there is a delete operation, the database just marks the data deleted. In case of updating the data, the database will make a copy of existing data, mark the data as deleted and then modify the existing entity.

This solves most of the problems regarding soft update and soft delete in case of single table. This approach does not provide a solution for going back in time. How do we find the parent of the current row? For this it is possible to use extra column that point to the parent row and the oldest row will have the pointer set to null. This means that when soft updating data it is necessary to not only change the payload, but the pointer to the parent row.

Apart from parent row pointer, the row must have meta fields like *CreatedAt* and *DeletedAt* for keeping track of which rows are active, and which are already deleted. Since we always copy the row before updating it, we need to make sure that the newly created rows have a unique primary key. This is achieved by having a composite key of *createdAt* and *rowId*. While we copy the parent row, we change the *deletedAt* to current time and update the existing rows *parentDeletedAt* to current date.

This pegs the question, how does soft update and soft delete work with other tables in scenario where we have one to many relationship. In terms of updating a child, everything remains the same as with one table, since we don't have any constraints that restricts master from having many children. Now if we want to update the master, then we repeat the same steps as before – copy the data and update the existing row. The foreign keys will be updated automatically if the foreign key has a constraint cascade update. This would be fine if there is no need to persist the relationships between the soft deleted master and children. If there is a need to persist the relationship information, then it is necessary to create a copy of all children that will point to the master. Soft Deleting, however would be easy, since the database will take care of updating foreign keys.

Now, how will soft delete work with a one to one relationship. The foreign key will be added just on the one side of the relationship and the one to one relationship is enforced by adding a constraint. Constraint states that all the foreign keys inside that table must be unique. When updating the master, then the copy of existing data must be done on both ends. This is needed in order for relationships to remain valid.

If the row, which contains the foreign key is soft deleted, then it is necessary to make the copy of the row to which the foreign key points to. This is needed, if there is a need to insert a new row that will point to existing foreign key. When we make a new copy of the row, then this will be avoided, since we created a row with a new composite primary key.

There are many ways to achieve the soft delete depending on the business case. In the current case the soft deletion and update is achieved by having composite key that consists of id and *deletedAt* row. Ability of moving back in time and seeing the older rows is achieved by having an extra column *ParentDeletedAt* which points to the *DeletedAt* column of the parent. Accompanying SQL script for achieving that will be found in extra 1 – soft delete and soft update.

Repository Pattern

Repository pattern is an architectural pattern where database logic is loosely coupled with business logic. This makes the code more testable and hence more maintainable in the long run. Other aspect the repository pattern removes is the code duplication. Business logic layer will only call the specified methods from the repository and won't repeat the same SQL queries inside the methods.

Repository pattern separates the data access logic and maps it to the entities in the business logic. It works with the domain entities and performs data access logic. Instead of writing SQL inside our logic, we will just call methods like *GetCustomerById*. Repository user is also not aware where and how the data is data and stored. 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.

So how does repository pattern compare to data access object. First of, quick overview of the DAO. Dao is a pattern that provides abstract interface to persistence layer. It's similar to repository pattern in the sense it abstracts away the specific details of the persistence layer. So what are the main differences – repository pattern has a specific repository for each entity and usually is concerned with operations done with single entity. A DAO on the other hand is a class that locates the data. The pattern doesn't restrict you to store data of the same type, thus you can easily have a DAO that locates/store related objects. So in conclusion they both abstract away the persistence layer while the DAO is more flexible and repository pattern is strictly talking about single entity.

This project is using repository pattern. Every domain object is mapped to a specific repository. There is a base repository inside DAL.Base.EF that is being inherited by every repository interface that is located inside Contracts.DAL.APP. Lastly the repositories that are used are inside DAL.App.EF which inherit from their specific repository interface. SO currently there are same amount of repositories as there is Domain entities.

Used materials

1. X-Tee [<https://moodle.ria.ee/mod/page/view.php?id=288>]
2. Liisa program [<https://medisoft.ee/tooted/haigla-infosusteeem-liisa/>]
3. Final thesis [https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=28941&year=2013]
4. Implementing soft delete on entity framework [<https://www.meziantou.net/entity-framework-core-soft-delete-using-query-filters.htm>]

Extra 1 – Soft delete and soft update

-- Using 1 : 01 Relationship with composite key

```
CREATE TABLE Students (  
    StudentId UNIQUEIDENTIFIER NOT NULL DEFAULT NEWID(),  
    Name VARCHAR(128) NOT NULL,  
    CreatedAt DATETIME2 NOT NULL DEFAULT getDate(),  
    DeletedAt DATETIME2 NOT NULL DEFAULT '9999-01-01',  
    PRIMARY KEY (StudentId, DeletedAt),  
)  
  
CREATE TABLE StudentImage (  
    StudentImageId UNIQUEIDENTIFIER NOT NULL DEFAULT NEWID(),  
  
    ImgURL VARCHAR(128) NOT NULL,  
    CreatedAt DATETIME2 NOT NULL DEFAULT getDate(),  
    DeletedAt DATETIME2 NOT NULL DEFAULT '9999-01-01',  
    PRIMARY KEY (StudentId, DeletedAt),  
  
    StudentId UNIQUEIDENTIFIER NOT NULL,  
    StudentDeletedAt DATETIME2 NOT NULL,  
    CONSTRAINT FK_StudentImage_Student FOREIGN KEY (StudentId, StudentDeletedAt)  
    REFERENCES Students(StudentId, DeletedAt) ON UPDATE CASCADE,  
    CONSTRAINT FK_StudentImage_Student_Unique UNIQUE(StudentId, StudentDeletedAt)  
)  
GO  
INSERT INTO Students (Name) VALUES ('Tristan')  
INSERT INTO Students (Name) VALUES ('Fido')  
INSERT INTO Students (Name) VALUES ('Ragnar')  
INSERT INTO Students (Name) VALUES ('Riin')  
GO  
USE trkras_hw2_softdelete  
GO  
  
INSERT INTO StudentImage (StudentImageId, ImgURL, StudentDeletedAt, StudentId) VALUES  
(NEWID(), 'www.tristan.ee', '9999-01-01', -- Select rows from a Table or View '[Students]' in schema '[dbo]'  
(SELECT StudentId FROM [dbo].[Students]  
WHERE Name = 'Tristan')  
)
```

```
INSERT INTO StudentImage (StudentImageId, ImgURL, StudentDeletedAt, StudentId) VALUES
(NEWID(),'www.fido.ee', '9999-01-01', (SELECT StudentId FROM [dbo].[Students]
WHERE Name = 'Fido'))
```

```
INSERT INTO StudentImage (StudentImageId, ImgURL, StudentDeletedAt, StudentId) VALUES
(NEWID(),'www.raku.ee', '9999-01-01', (SELECT StudentId FROM [dbo].[Students]
WHERE Name = 'Ragnar'))
```

```
INSERT INTO StudentImage (StudentImageId, ImgURL, StudentDeletedAt, StudentId) VALUES
(NEWID(),'www.riin.ee', '9999-01-01', (SELECT StudentId FROM [dbo].[Students]
WHERE Name = 'Riin'))
```

```
-- Updating students table
-- Step 1) Copy the old object and change DeletedAt to getDate()
-- Step 2) Change the desired column value
```

```
INSERT INTO Students (StudentId, Name, CreatedAt, DeletedAt)
SELECT StudentId, Name, CreatedAt, getDate() FROM Students
WHERE StudentId = (SELECT StudentId FROM Students
WHERE Name = 'Tristan')
GO
```

```
UPDATE Students
SET Name = 'Tristan2'
WHERE DeletedAt = '9999-01-01' AND StudentId = (SELECT StudentId FROM Students
WHERE Name = 'Tristan' AND DeletedAt = '9999-01-01')
GO
```

```
-- Soft Delete rows from table '[Students]' in schema '[dbo]'
-- Step 1) Since there is a update cascade constraint that after
-- updating the master the child FK-s will be updated automatically.
-- Step 2) Manually set the child delted at to correct date
```

```
DECLARE @NowTime DATETIME2
SELECT @NowTime = getDate()
UPDATE Students
SET DeletedAt = @NowTime
WHERE DeletedAt = '9999-01-01' AND StudentId = (SELECT StudentId FROM Students
WHERE Name = 'Tristan2' AND DeletedAt = '9999-01-01')
UPDATE StudentImage
SET DeletedAt = @NowTime
```

```

WHERE DeletedAt = '9999-01-01' AND StudentId = (SELECT StudentId FROM Students
WHERE Name = 'Tristan2' AND DeletedAt = @NowTime)
GO

```

```

-- Drop existing constraint
-- Create new constraint

```

```

ALTER TABLE StudentImage
    DROP CONSTRAINT FK_StudentImage_Student

```

```

ALTER TABLE StudentImage
    ADD CONSTRAINT FK_StudentImage_Student FOREIGN KEY (StudentId, StudentDeletedAt)
    REFERENCES Students(StudentId, DeletedAt) ON UPDATE CASCADE

```

```

INSERT INTO Students (StudentId, Name, CreatedAt, DeletedAt)

```

```

SELECT StudentId, Name, CreatedAt, @NowTime FROM Students
WHERE StudentId = (SELECT StudentId FROM Students
WHERE Name = 'Tristan')
GO

```

```

-- What happens if we delete the child entity itself.
-- We first make a copy of the parent
-- Then we just soft delete aswell and point the child to the old record.

```

```

DECLARE @NowTime DATETIME2
SELECT @NowTime = getDate()

```

```

UPDATE Students
SET DeletedAt = @NowTime
WHERE DeletedAt = '9999-01-01' AND StudentId = (SELECT StudentId FROM Students
WHERE Name = 'Ragnar' AND DeletedAt = '9999-01-01')

```

```

UPDATE StudentImage
SET DeletedAt = @NowTime
WHERE DeletedAt = '9999-01-01' AND StudentId = (SELECT StudentId FROM Students
WHERE Name = 'Ragnar' AND DeletedAt = @NowTime)

```

```

INSERT INTO Students (StudentId, Name, CreatedAt, DeletedAt)

```

```

SELECT StudentId, Name, CreatedAt, '9999-01-01' FROM Students
WHERE DeletedAt = @NowTime AND StudentId = (SELECT StudentId FROM Students
WHERE Name = 'Ragnar' AND DeletedAt = @NowTime)
GO

```

-- Selecting data that is not deleted from 1 table

```

SELECT 'Correct Data after soft delete'
SELECT *
FROM Students
WHERE CreatedAt <= getDate() AND (DeletedAt IS NULL OR DeletedAt > getDate())

```

-- Selecting data that is not deleted from 2 tables

```

SELECT 'Correct Data after soft delete'
SELECT s.StudentId, s.Name, s.CreatedAt, s.DeletedAt, sm.ImgURL, sm.CreatedAt, sm.DeletedAt
FROM Students AS s
LEFT JOIN StudentImage AS sm ON s.StudentId = sm.StudentId AND
sm.StudentDeletedAt = s.DeletedAt
WHERE s.DeletedAt = '9999-01-01' AND sm.DeletedAt = '9999-01-01'
GO

```

```

SELECT * FROM Students
SELECT * FROM StudentImage

```

-- So now the same solution for 1:M relationship in DB

```

USE trkras_hw2_softdelete
GO
DROP TABLE dbo.Person;
GO

```

```

CREATE TABLE [dbo].[Person] (
    PersonId    UNIQUEIDENTIFIER    NOT NULL    default NEWID(),
    FirstName   VARCHAR(128)    NOT NULL,
    LastName    VARCHAR(128)    NOT NULL,
    IdCode      VARCHAR(25),

    CreatedAt   DATETIME2 NOT NULL DEFAULT getDate(),
    DeletedAt   DATETIME2 NOT NULL DEFAULT '9999-01-01',
    PRIMARY KEY (PersonId, DeletedAt),

```

```
);
```

```
GO
```

```
CREATE TABLE [dbo].[Appointment] (  
    AppointmentId    UNIQUEIDENTIFIER    NOT NULL    default NEWID(),  
    AppointmentName  VARCHAR(128)    NOT NULL,  
    AppointmentLength INTEGER NULL,  
    StartingTime    DATETIME2 NOT NULL,  
    EndingTime      DATETIME2 NULL,  
  
    CreatedAt DATETIME2 NOT NULL DEFAULT getDate(),  
    DeletedAt DATETIME2 NOT NULL DEFAULT '9999-01-01',  
    PersonId   UNIQUEIDENTIFIER NOT NULL,  
    PersonDeletedAt DATETIME2 NOT NULL,  
  
    PRIMARY KEY (AppointmentId, DeletedAt),  
    CONSTRAINT FK_Appointment_Person FOREIGN KEY (PersonId, PersonDeletedAt)  
    REFERENCES Person(PersonId, DeletedAt) ON UPDATE CASCADE,
```

```
);
```

```
GO
```

```
DROP TABLE Appointment;
```

```
GO
```

```
INSERT INTO Person (FirstName, LastName, IdCode) VALUES ('Tristan', 'Krass', '39806302730')  
INSERT INTO Person (FirstName, LastName, IdCode) VALUES ('Katrin', 'Viira', '4972102012')  
INSERT INTO Person (FirstName, LastName, IdCode) VALUES ('Karmen', 'Kass', '4757801248421')  
INSERT INTO Person (FirstName, LastName, IdCode) VALUES ('Reena', 'Kuup', '4824178712')
```

```
GO
```

```
-- Delete rows from table '[Person]' in schema '[dbo]'
```

```
DELETE FROM [dbo].[Person]
```

```
WHERE FirstName = 'Reena'
```

```
GO
```

```
SELECT * FROM Person
```

```
SELECT * FROM [dbo].[Appointment]
```

```
INSERT INTO [dbo].[Appointment](AppointmentName, AppointmentLength, StartingTime, EndingTime, PersonId,  
PersonDeletedAt)
```

```
VALUES ('Visiit1', 45, '2020-03-07 12:00', '2020-03-07 12:45',
```

```
(SELECT PersonId FROM [dbo].[Person]
```

```

WHERE FirstName = 'Reena' AND DeletedAt = '9999-01-01'),
(SELECT DeletedAt FROM [dbo].[Person]
WHERE FirstName = 'Reena' AND DeletedAt = '9999-01-01'))

```

```

INSERT INTO [dbo].[Appointment](AppointmentName, AppointmentLength, StartingTime, EndingTime, PersonId,
PersonDeletedAt)
VALUES ('Visiit2', 45, '2020-03-08 12:00', '2020-03-07 12:45',
(SELECT PersonId FROM [dbo].[Person]
WHERE FirstName = 'Tristan' AND DeletedAt = '9999-01-01'),
(SELECT DeletedAt FROM [dbo].[Person]
WHERE FirstName = 'Tristan' AND DeletedAt = '9999-01-01'))

```

```

INSERT INTO [dbo].[Appointment](AppointmentName, AppointmentLength, StartingTime, EndingTime, PersonId,
PersonDeletedAt)
VALUES ('Visiit2', 60, '2020-03-09 12:00', '2020-03-07 13:00',
(SELECT PersonId FROM [dbo].[Person] WHERE FirstName = 'Tristan3' AND DeletedAt = '9999-01-01'),
(SELECT DeletedAt FROM [dbo].[Person]
WHERE FirstName = 'Tristan3' AND DeletedAt = '9999-01-01'))

```

```

INSERT INTO [dbo].[Appointment](AppointmentName, AppointmentLength, StartingTime, EndingTime, PersonId,
PersonDeletedAt)
VALUES ('EMO', 40, '2020-03-09 10:00', '2020-03-07 10:30',
(SELECT PersonId FROM [dbo].[Person] WHERE FirstName = 'Tristan3' AND DeletedAt = '9999-01-01'),
(SELECT DeletedAt FROM [dbo].[Person]
WHERE FirstName = 'Tristan3' AND DeletedAt = '9999-01-01'))

```

```

INSERT INTO [dbo].[Appointment](AppointmentName, AppointmentLength, StartingTime, EndingTime, PersonId,
PersonDeletedAt)
VALUES ('Visiit3', 45, '2020-03-08 13:00', '2020-03-07 13:45',
(SELECT PersonId FROM [dbo].[Person]
WHERE FirstName = 'Karmen' AND DeletedAt = '9999-01-01'),
(SELECT DeletedAt FROM [dbo].[Person]
WHERE FirstName = 'Karmen' AND DeletedAt = '9999-01-01'))

```

```

INSERT INTO [dbo].[Appointment](AppointmentName, AppointmentLength, StartingTime, EndingTime, PersonId,
PersonDeletedAt)
VALUES ('Visiit4', 45, '2020-03-08 14:00', '2020-03-07 14:45',
(SELECT PersonId FROM [dbo].[Person]

```

```

WHERE FirstName = 'Katrin' AND DeletedAt = '9999-01-01'),
(SELECT DeletedAt FROM [dbo].[Person]
WHERE FirstName = 'Katrin' AND DeletedAt = '9999-01-01'))

```

```

-- What happens when you delete the child?
-- Nothing, we will just let the child to have
-- the foreign key since there is no problem with SQL

```

```

UPDATE [dbo].[Appointment]
SET DeletedAt = getDate()
WHERE AppointmentName = 'Visiit1' AND AppointmentLength = 45
GO

```

```

-- No problem when inserting new appointments to the same person

```

```

INSERT INTO [dbo].[Appointment](AppointmentName, AppointmentLength, StartingTime, EndingTime, PersonId,
PersonDeletedAt)
VALUES ('Visiitx', 45, '2020-03-07 17:00', '2020-03-07 12:45',
(SELECT PersonId FROM [dbo].[Person]
WHERE FirstName = 'Reena' AND DeletedAt = '9999-01-01'),
(SELECT DeletedAt FROM [dbo].[Person]
WHERE FirstName = 'Reena' AND DeletedAt = '9999-01-01'))

```

```

-- What happens if we soft delete the person ID
-- Since we have cascade update, the db will
-- take care of updating the FK of children aswell.
-- only thing left, is to decide whether or not to update the
-- DeletedAt on children aswell. I decide to do that.

```

```

DECLARE @DAT DATETIME2
SELECT @DAT = getDate()

```

```

UPDATE [dbo].[Person]
SET DeletedAt = @DAT
WHERE PersonId = (SELECT PersonId FROM Person WHERE FirstName = 'Reena' AND DeletedAt = '9999-01-01') AND DeletedAt = '9999-01-01'

```

```

UPDATE [dbo].[Appointment]
SET DeletedAt = @DAT
WHERE DeletedAt = '9999-01-01' AND PersonId =

```



```

(SELECT PersonId FROM Person WHERE FirstName = 'Reena' AND DeletedAt = @DAT)
GO

-- Doing Joins and querying data
SELECT 'All the data from persons'
SELECT * FROM Person -- All the data available
SELECT 'All the data from Appointments'
SELECT * FROM Appointment -- All the Appointment data
SELECT 'All the data from persons that are not deleted'
SELECT * FROM Person WHERE DeletedAt >= getDate() AND CreatedAt <= getDate() -- Only rows that are
available
SELECT 'All the data from appointments that are not deleted'
SELECT * FROM Appointment WHERE DeletedAt >= getDate() AND CreatedAt <= getDate() -- Only rows that
are available

-- Making joins
SELECT 'All the data from persons and appointments are not deleted'
SELECT p.PersonId, p.FirstName, p.IdCode, a.AppointmentName,a.DeletedAt, a.PersonId, a.AppointmentId
FROM Person AS p
LEFT JOIN Appointment AS a
ON p.PersonId = a.PersonId AND p.DeletedAt = a.PersonDeletedAt
WHERE a.DeletedAt >= getDate() AND a.CreatedAt <= getDate() AND
p.DeletedAt >= getDate() AND p.CreatedAt <= getDate()
GO

-- Updating the master is the exact same process as 1:1 relationship
-- When Updating the master we will also have to update the children
-- So instead of 1 soft-update we have to do as many soft-updates
-- as many children the master has
INSERT INTO Person (PersonId, FirstName,LastName,IdCode, CreatedAt, DeletedAt)
SELECT PersonId, FirstName, LastName, IdCode, CreatedAt, getDate() FROM Person
WHERE DeletedAt = '9999-01-01' AND PersonId = (SELECT PersonId FROM Person
WHERE FirstName = 'Tristan' AND DeletedAt = '9999-01-01')

UPDATE Person
SET FirstName = 'Tristan2'
WHERE DeletedAt = '9999-01-01' AND PersonId = (SELECT PersonId FROM Person
WHERE FirstName = 'Tristan' AND DeletedAt = '9999-01-01')
GO

-- Second update

```

```

INSERT INTO Person (PersonId, FirstName, LastName, IdCode, CreatedAt, DeletedAt)
SELECT PersonId, FirstName, LastName, IdCode, CreatedAt, getDate() FROM Person
WHERE DeletedAt = '9999-01-01' AND PersonId = (SELECT PersonId FROM Person
WHERE FirstName = 'Tristan2' AND DeletedAt = '9999-01-01')

```

```

UPDATE Person
SET FirstName = 'Tristan3', LastName = 'Krass3'
WHERE DeletedAt = '9999-01-01' AND PersonId = (SELECT PersonId FROM Person
WHERE FirstName = 'Tristan2' AND DeletedAt = '9999-01-01')
GO

```

```

INSERT INTO Appointment (AppointmentId, AppointmentName, AppointmentLength, StartingTime, EndingTime,
CreatedAt, DeletedAt, PersonId, PersonDeletedAt)
SELECT AppointmentId, AppointmentName, AppointmentLength, StartingTime, EndingTime,
CreatedAt, getDate(), PersonId, PersonDeletedAt
FROM Appointment
WHERE DeletedAt = '9999-01-01' AND
AppointmentId = 'ccfbcffc-fa24-44e2-b264-6e0f9b66d125'
AND PersonId = (SELECT PersonId FROM Person
WHERE FirstName = 'Tristan3' AND DeletedAt = '9999-01-01')

```

```

UPDATE Appointment
SET AppointmentName = 'Erakorraline2222'
WHERE DeletedAt = '9999-01-01' AND
PersonId = (SELECT PersonId FROM Person
WHERE FirstName = 'Tristan3' AND DeletedAt = '9999-01-01')

```

```

-- Get the history of Persons
SELECT * FROM Person
WHERE CreatedAt = '2020-03-07 21:30:27.0333333'
GO

```

```

-- Get the history of Appointments, since the createdAt timestamp will not change
SELECT * FROM Appointment
WHERE CreatedAt = '2020-03-07 21:38:35.9400000'
GO

```

```

-- From here we can just inside the code make the history

```

```

-- Since most of the time we actually just want the data

```

-- that is not created it's easier to create a view

```
CREATE VIEW [ActivePersons] AS
SELECT PersonId, FirstName, LastName, IdCode, CreatedAt, DeletedAt
FROM Person
WHERE DeletedAt >= getDate() AND CreatedAt <= getDate()
GO
```

```
SELECT * FROM ActivePersons
```

-- Adding A unique column that points to the previous

```
CREATE TABLE Dummy (
    DummyId    UNIQUEIDENTIFIER    NOT NULL    default NEWID(),
    DummyName  VARCHAR(128)    NOT NULL,

    CreatedAt  DATETIME2 NOT NULL DEFAULT getDate(),
    DeletedAt  DATETIME2 NOT NULL DEFAULT '9999-01-01',

    ParentDeletedAt  DATETIME2 NULL,
    -- Cannot add a constraint for UNIQUE Parent, since parent has always null as A parent
    -- CONSTRAINT Dummy_Parent_Unique UNIQUE(ParentDummyId, ParentDeletedAt),
    CONSTRAINT PK_Dummy_DAT PRIMARY KEY (DummyId, DeletedAt)
)
GO
```

```
ALTER TABLE Dummy
DROP CONSTRAINT Dummy_Parent_Unique
GO
```

```
DROP TABLE Dummy
GO
```

```
INSERT INTO Dummy (DummyName) VALUES ('Foo')
GO
```

```
DECLARE @DAT DATETIME2
SELECT @DAT = getDate()
```

```
INSERT INTO Dummy (DummyId, DummyName, CreatedAt, DeletedAt, ParentDeletedAt)
SELECT      DummyId, DummyName, CreatedAt, @DAT,    ParentDeletedAt FROM Dummy
```

```
WHERE DummyId = 'fce09c4d-e040-4262-b745-f6f55a677d40' AND DeletedAt = '9999-01-01'
```

```
UPDATE Dummy
```

```
SET DummyName = 'Bar', ParentDeletedAt = @DAT
```

```
WHERE DummyId = 'fce09c4d-e040-4262-b745-f6f55a677d40' AND DeletedAt = '9999-01-01'
```

```
GO
```

```
DECLARE @DAT DATETIME2
```

```
SELECT @DAT = getDate()
```

```
INSERT INTO Dummy (DummyId, DummyName, CreatedAt, DeletedAt, ParentDeletedAt)
```

```
SELECT DummyId, DummyName, CreatedAt, @DAT, ParentDeletedAt FROM Dummy
```

```
WHERE DummyId = 'fce09c4d-e040-4262-b745-f6f55a677d40' AND DeletedAt = '9999-01-01'
```

```
UPDATE Dummy
```

```
SET DummyName = 'Fido', ParentDeletedAt = @DAT
```

```
WHERE DummyId = 'fce09c4d-e040-4262-b745-f6f55a677d40' AND DeletedAt = '9999-01-01'
```

```
GO
```

```
SELECT * FROM Dummy
```

```
-- Now we can travel back in time
```

```
-- Get parent
```

```
SELECT * FROM Dummy AS ParentChild WHERE DeletedAt = (
```

```
SELECT ParentDeletedAt FROM Dummy
```

```
WHERE DummyId = 'fce09c4d-e040-4262-b745-f6f55a677d40' AND DeletedAt = '9999-01-01')
```

```
SELECT * FROM Dummy WHERE DeletedAt = (SELECT ParentDeletedAt FROM Dummy WHERE DeletedAt =  
(
```

```
SELECT ParentDeletedAt FROM Dummy
```

```
WHERE DummyId = 'fce09c4d-e040-4262-b745-f6f55a677d40' AND DeletedAt = '9999-01-01'))
```

```
SELECT * FROM Dummy WHERE parentDeletedAt IS NOT NULL
```