Tallinn University of Technology
School of Information Technologies

Artjom Pahhomov

School Management System

Distributed Systems Project

Supervisor: Andres Käver

Tallinn 2020

1. Introduction

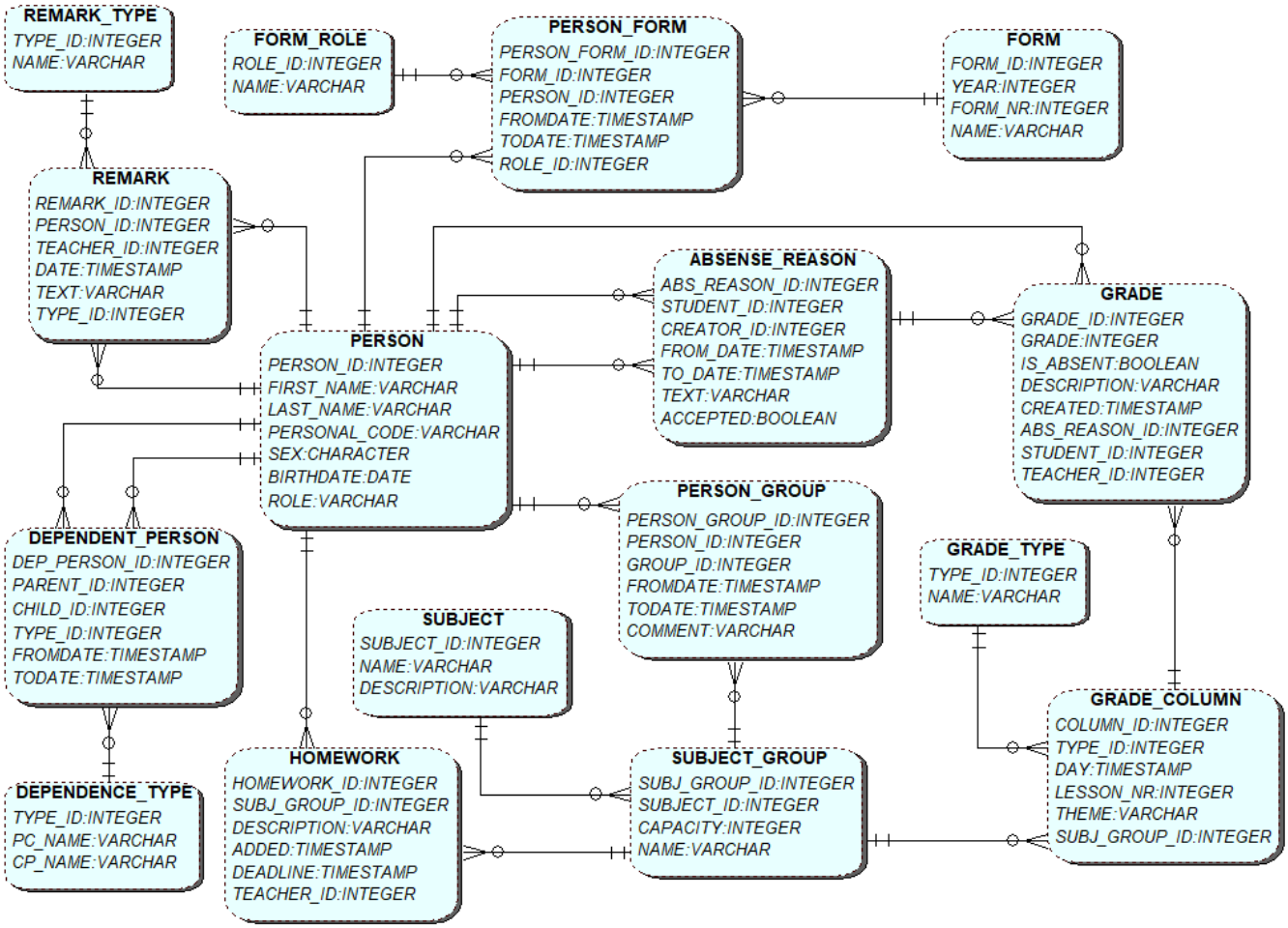
The goal of this project is to build a usable system for school management. This system will contain all the crucial and necessary functionality for ordinary school: a convenient and modern-looking grade book, the ability for teachers to post remarks, homework assignments, plan the lessons, add lesson themes, check attendance, and so on.

As for today, there are some popular school management systems in Estonia – most important of them are eKool and Stuudium, which are used by an overwhelming majority of schools – but in author's opinion, each of those systems has its flaws and specific features that make those systems hard to use for ordinary student, parent or teacher. For everyone, systems with intuitive design and usability are equated with easy-to-use systems, and thus the author finds the lack of proper design disturbing. Intuitive design is the key to the popularity of a certain software product among the users.

The second big problem of those systems is that there is a lot of functionality no one has ever even considered to use. There is no point in adding rudimentary functionality since it only increases the complexity but has no effect on the users of the given system. According to some studies, up to 40% of features in any software product are never used. The reason why it is decided to take on this project. The plan is to mainly concentrate on the crucial functions that will certainly be used. The main goal the author is planning to achieve is to create a convenient and intuitive system with the functionality users really need.

The scope of this project is to create a minimum viable product suitable for most schools in Estonia. The author's motivation is to build a system that is indeed easy-to-use, a system that has an intuitive design and only necessary features. The system itself is being built using .NET Core tools.

2. Entity Relationship Diagram



3. Analysis and implementation of soft-update and soft-delete

In this part of the documentation, the author intends to go through certain technical aspects of relational databases. The subject of this paragraph is designing and implementing a SQL database for auditable/reversible data changes. This technology is also known as soft-update and soft-delete.

3.1. Single table

Designing a soft-delete and soft-update solution for a single table is a relatively easy problem. The author came up with a solution to using composite primary key – identification number and deletion time. This solution helps with binding all the related data together (the identification number does not change, only deletion time is changed) without the violation of the primary key's uniqueness. This solution, however, has its cons, because now the author is fated to always use some date as deletion time. Reason being the fact that the primary key or its parts cannot be nullable. The order of operations is the following:

- Soft-update:
    1. insert a copy of a record, but with the current time as deletion time;
    2. cascade update the record with the new information and last edit time.
- Soft-delete:
    1. set the deletion time to the current time.

3.2. One-to-many relationship

In order to solve the problem of updating the primary key and breaking all the possible joins, the author decided to use a composite primary key. The primary keys of both tables consist of identification number (also known as ID) and an additional field that contains the deletion time. This approach is also used in the next paragraph. Furthermore, when creating tables, the foreign key constraint of many-side was marked to cascade update. This means the referencing rows are updated in the child table when the referenced row is

updated in the parent table which has a primary key. So, taking into consideration all the aspects, the author came up with the following order of operations:

- Soft-update *one*-side:
    1. insert a copy of a record, but with the current time as deletion time;
    2. add a copy of all the dependent records in the *many*-table, also updating deletion time of them and foreign key (deletion time – part of a composite primary key of *one*-table);
    3. cascade update the record with the new information and last edit time.
- Soft-delete *one*-side:
    1. cascade update the record with the new deletion time;
    2. manually update the dependent records with the new deletion time.

Although, when soft-deleting data from the many-side, it is possible to save the state of data at the one-side, author decided not to implement this because, in author's point of view, data at the many-side can change too often, and it could have caused table at the one-side of the relation to containing a lot of basically same records. It is still possible to return data as it was in each moment of time because one-side does not contain the time of creation/deletion of the many-side. So, soft-update and soft-delete on the many-side look the same as described in paragraph 3.1.

3.3. One-to-zero/one relationship

This relationship turned out to be the hardest to implement the soft-delete and soft-update on, nevertheless, it was not impossible. A solution that the author came up with was to add the unique constraint to zero/one-side which consists of three columns: identification number of the parent, deletion time of the parent (being the foreign key of a parent) and deletion time of the child. The latter helps by pointing at the presence of a valid child, so the author gets rid of multiple problems. For example, if a new parent table already has a deleted child, then a new child can still be added because although the combination of deletion time and ID of the parent is not unique, the deletion time of the child adds the needed uniqueness. If the valid child is already present at the table, one cannot add more children because of the unique constraint.

- Soft-update *one*-side:
    1. insert a copy of a record, but with the current time as deletion time;
    2. insert a copy of all the dependent records in the *zero/one*-table, also updating deletion time of them and foreign key;
    3. cascade update the record with the new information and last edit time.
- Soft-delete *one*-side:
    1. cascade update the record with the new deletion time;
    2. update the dependent records with the new deletion time.
- Soft-update *zero/one*-side:
    1. insert a copy of a record, but with the current time as deletion time;
    2. update the record with the new information and last edit time.
- Soft-delete *zero/one*-side:
    1. set the deletion time to current time.

In this case, the author also has decided in favor of not saving the state on one-side every time the zero/one-side is changed. Accessing the state of the database in each moment can still be done because the one-side does not contain information about records on zero/one-side.