TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Raul Vesinurm 190510IADB

INIMESELT INIMESELE TÖÖRIISTADE RENTIMISE PLATVORMI LOOMINE

Iseseisev töö

Juhendaja: Andres Käver

Tallinn 2020

Autorideklaratsioon

Kinnitan, et olen koostanud antud iseseisva töö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud.

Autor: Raul Vesinurm

23.02.2020

Sisukord

1 Sissejuhatus

Soovin luua keskkonda, kus inimesed saavad rentida üksteiselt tööriistu, tehnikat jne. Kellel meist ei seisaks kuuris mõni tööriist, mida kasutame aastas ainult mõne korra. Miks mitte seda välja rentida ja raha teenida? Inimesed, kes ei soovi või pole võimalust soovitud asja soetada, saavad seda rentida meie veebikeskkonnas. Anname võimaluse ka ettevõtetele enda toodete postitamiseks meie veebilehele. Sellega saavad ettevõtjad lisaks ka reklaamida ennast rohkem.

2 Analüüs

Meil on firma, mis vahendab inimeselt inimesele tööriistarendi teenust. Me oleme vahendajateks tööriistade omanike ja rentijate vahel.

Meie keskkonna kasutamiseks tuleb kliendil luua kasutajakonto. Konto aktiveerimiseks kasutame SendGrid teenust, kus registreeritud kasutaja emailile saadetakse kinnituskiri, milles on link, mis aktiveerib konto. Kasutajal on võimalus lisada tooteid, mida ta soovib välja rentida teistele kasutajatele. Kasutaja saab rentida tooteid, mida teised kasutajad on rentimiseks välja pannud. Kui kasutaja soovib asju rentida või rendile anda enda ettevõtte alt, siis on tal selleks võimalus olemas. Iga kasutaja saab lisada süsteemi enda ettevõtte, millega ta seotud on. Kasutajal on võimalus näha, mida iga konkreetne temaga seotud firma rendib ja on rentinud teistelt kasutajatelt.

Kasutajate poolt lisatud esemetel on kirjeldus, mis tootega tegemist, toote mark ja mudel. Esemele on võimalik lisada pilte. Ese saab kuuluda mitmesse kategooriasse. Esemele määratakse asukoht, kus see asub. Kasutaja peab ära määrama, millised on transpordi võimalused eseme kättesaamiseks (kuller, pakiautomaat, tulen ise järele). Tootele saab lisada toodet iseloomustavaid väärtusi (toote kaal, mõõtmed, võimsus). Kasutajatel on võimalus näha konkreetse kasutaja/firma kõiki pakutavaid tooteid. Kasutaja peab toote rentimiseks valima kuupäevad, millal tal konkreetset eset vaja läheb. Meie süsteem peab kasutajatele näitama kuupäevasid, millal on toode vaba ja millal mitte. Kasutaja määrab igale tootele hinna vastavalt rendiperioodile. Rendiperioodideks on päev, nädal ja kuu. Rendiperiood ’päev’ rakendatakse siis, kui eset renditakse alla 7 päeva. Rendiperiood ’nädal’ rakendatakse siis, kui eset renditakse vahemikus 7 – 30 päeva. Mida kauemaks eset renditakse, seda odavam rendi päevahind tuleb.

Kui kasutaja on leidnud endale vajamineva eseme, valinud kuupäevad, sobiva transpordiliigi, siis seejärel saab toote lisada ostukorvi. Kasutajal on võimalus veel teisigi tooteid ostukorvi lisada või siis eseme/esemete eest tasuda. Peale edukat makse sooritust genereeritakse kasutajale arve.

Esialgu üritan loetletud funktsionaalsused teoks teha. Lisafunktsioonideks oleks näiteks kommentaaride lisamine toodete juurde, toote tagastamine(kui rendiperiood läbi).

3 Analysis of soft delete and soft update in SQL

3.1 Introduction

Any relationship requires that the parent table has a primary key (abv. PK) that uniquely identifies each row, and the child table has a foreign key (abv. FK) column that must be populated with values that are the same as some existing values of the primary key in the parent table. Foreign key value has to be unique in parent table, otherwise there would be no relational data integrity.

In one to many (1:m) relationship the foreign key value can be repeated in child table as many times as needed. For example, one person can have several contacts.

A one to zero or one (1:0-1) relationship exists when only one row of data in the principal table is linked to zero or one row in a dependent table. In order to add data to dependent table then firstly, data needs to be inserted to the principal table and then a row can be added to the dependent table. That is because foreign key in dependent table is not optional. In 1:0-1 relationship the foreign key itself must be a unique in the child table. This can be achieved by making unique constraint that tells to the database which field has to be unique in the entire table. UNIQUE keyword is used for that. FK should only be in child table. If FK has been put to both sides, then it would be two different relationships. 1:m relationship has to be considered to define where FK is put.

What differentiates a one-to-one relationship from a one-to-many relationship purely in terms of implementation is whether the child table's foreign key value has a Unique or Primary Key constraint.

Primary Key is used to identify rows uniquely in a table. A PRIMARY KEY constraint automatically has a UNIQUE constraint. Primary key cannot be NULL, because primary key’s purpose is to uniquely identify records. If two records of a single column have a NULL value, the column values are not considered equal. In simple words two

NULL values are not considered as equal. Primary Key cannot have NULL values as they are not compared with any other value.

A foreign key is a key used to link two tables together. A foreign key is a field in one table that refers to the primary key in another table. Foreign key can be NULL if relationship is done as optional.

To ensure that join between the parent and the child tables can be done, primary key cannot be NULL. If both PK and FK were NULL, then join would be impossible.

3.2 Soft update and delete

For soft update a copy of the old record needs to be made to keep the previous data. A simple copy, that creates two records with the same primary key, would be impossible, as the primary key has to be unique in the entire table. A composite key would be the solution. A composite key is a combination of two or more columns in a table that can be used to uniquely identify each row in the table. When the columns are combined, uniqueness is guaranteed, but when they are taken individually, uniqueness is not guaranteed. As the date of deletion needs to be marked down, it can be used as the second attribute in the composite key. This way, the composite key consists of Id and DeletedOn attributes. When the first record is created, DeletedOn date is set to be 9999-12-31. NULL cannot be used because primary key is not nullable. In Entity Framework DateTime.MaxValue is used. When making a copy of the record, DeletedOn date is set to the current date of when the updating is done. That guarantees that primary key will be unique. Next step would be to change old record's CreatedOn date to current date and data as well.

In the case of soft delete the DeletedOn date is set to current date. UPDATE statement is being used. It is important that the integrity of the referential key is not lost. When the primary key in the parent table is changed, then the foreign key in child table has to change as well. This can be done by including ON UPDATE CASDADE into foreign key constraint in the child table. ON UPDATE CASCADE means that if the parent primary key is changed, the child value will also change.

Soft update causes a problem in 1:0-1 relationship in the dependent table as the foreign key has to be unique. As only one active record can point to a record in the parent table, UNIQUE constraint has to be used. This UNIQUE constraint contains three columns – child table's DeletedOn, Foreign Key's ID and Foreign Key's DeletedOn. When doing soft update, Foreign Key ID and Foreign Key's DeletedOn fields always stay the same and child table's DeletedOn changes.

4 Design patterns

4.1 Introduction

Applications that need access to data can be implemented in very easy way. The simplest way is to write all the data access related code in the controllers. That way controllers are directly interacting with the Entity Framework data context class and execute the queries to retrieve the data from the database. Controllers also perform the INSERT, UPDATE, and DELETE operations using the data context and DbSet. The Entity Framework in turn talks with the underlying SQL database. This implementation works, but it suffers from the drawback that the database access code (i.e. creating the data context object, writing the queries, manipulating the data, persisting the changes to the database, etc.) is embedded directly inside the controller action methods. This design or implementation can cause code duplication and further, we need to change the controller even if we do a small change in the data access logic.

4.2 Repository pattern

Repository Design Pattern is used to create an abstraction layer between the data access layer and the business logic layer of the application. That abstraction layer is generally called the Repository Layer and it will directly communicate with the data access layer, gets the data and provides it to the business logic layer. That way controllers won't talk with the Entity Framework data context class directly and there is no queries or any other data access code in the action methods. The main advantage to use the repository design pattern is to isolate the data access logic and business logic. If we have to change something in data layer then we have to change only in one place (in repository) not all the controllers. Another benefit is that testing controllers becomes easy because the testing framework need not run against the actual database access code. All CRUD operations are wrapped by the repository. The repository uses the Entity Framework data context class to perform the CRUD operations.

4.3 Data transfer object

Data Transfer Objects are used to transfer data between the application layer and the presentation layer. DTO is a object for moving data from controller to view. Presentation layer is completely isolated from the domain layer. In an ideally layered application, the presentation layer never works with domain objects. The idea is not to show all the info from domain model to the user, only the info what is necessary.

Controller is responsible for manipulating business objects to retrieve the data to display to the user in a View. Since the typical View requires data from several business entities, you need to combine data from several sources into a Data Transfer Object to pass the data from the Controller to the View.

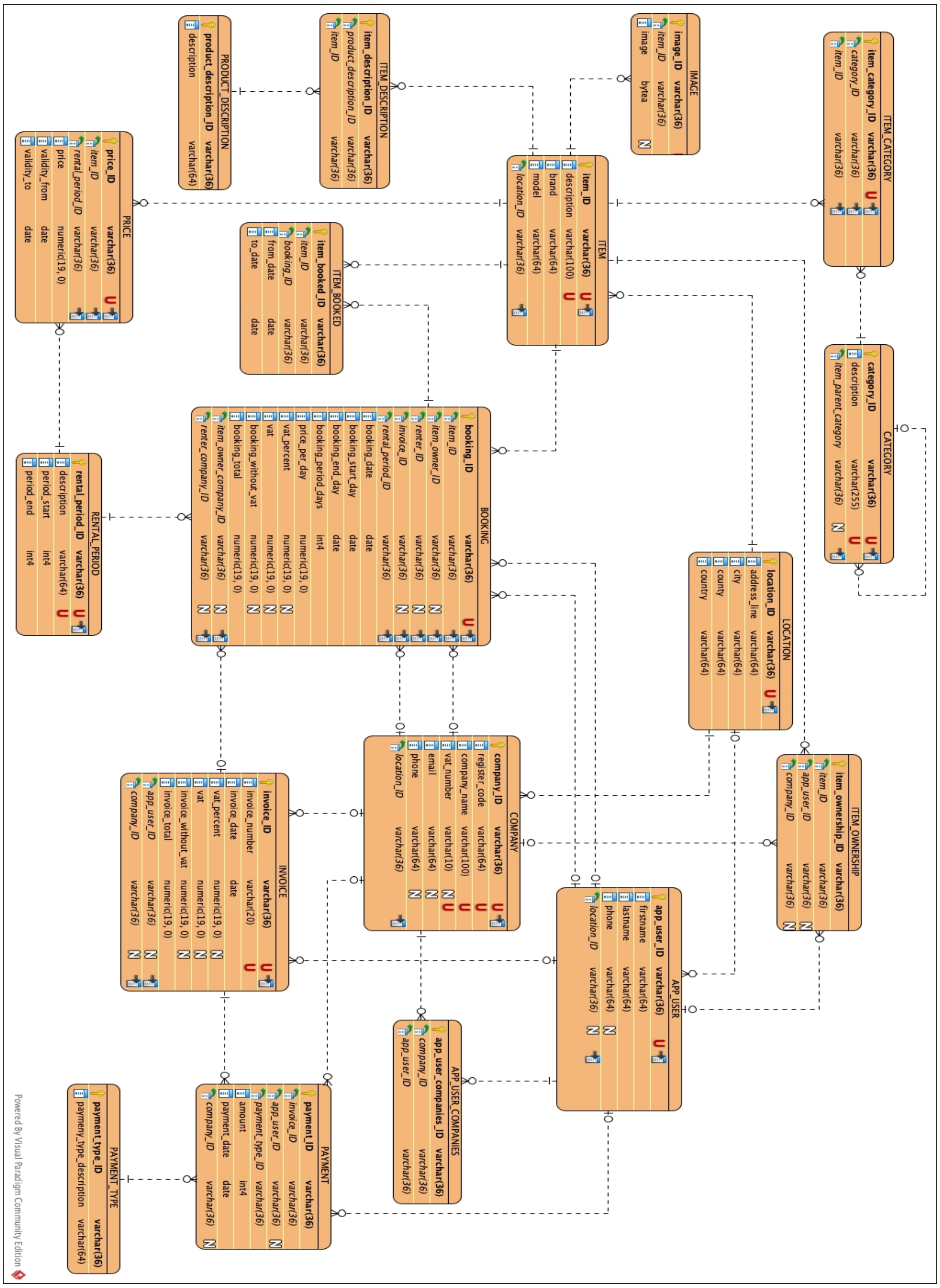
4.4 Conclusion

Design Pattern is used to create an abstraction layer between the data access layer and the application layer. DP idea is to separate data Access from business logic. DTO deals with data transfering between business and presentation layer. It eliminates that we do not use domain models in views.

Kasutatud kirjandus

There are no sources in the current document.

Lisa 1 – ERD skeem


PRODUCT_DESCRIPTION
product_description_ID varchar(36)
description varchar(64)
ITEM_DESCRIPTION
item_description_ID varchar(36)
product_description_ID varchar(36)
item_ID varchar(36)
IMAGE
image_ID varchar(36)
item_ID varchar(36)
image bytea
ITEM_CATEGORY
item_category_ID varchar(36)
category_ID varchar(36)
item_ID varchar(36)
PRICE
price_ID varchar(36)
item_ID varchar(36)
rental_period_ID varchar(36)
price numeric(19, 0)
validity_from date
validity_to date
ITEM_BOOKED
item_booked_ID varchar(36)
item_ID varchar(36)
booking_ID varchar(36)
from_date date
to_date date
ITEM
item_ID varchar(36)
description varchar(100)
brand varchar(64)
model varchar(64)
location_ID varchar(36)
CATEGORY
category_ID varchar(36)
description varchar(255)
item_parent_category varchar(36)
BOOKING
booking_ID varchar(36)
item_ID varchar(36)
item_owner_ID varchar(36)
renter_ID varchar(36)
invoice_ID varchar(36)
rental_period_ID varchar(36)
booking_date date
booking_start_day date
booking_end_day date
booking_period_days int4
price_per_day numeric(19, 0)
vat_percent numeric(19, 0)
vat numeric(19, 0)
booking_without_vat numeric(19, 0)
booking_total numeric(19, 0)
item_owner_company_ID varchar(36)
renter_company_ID varchar(36)
RENTAL_PERIOD
rental_period_ID varchar(36)
description varchar(64)
period_start int4
period_end int4
LOCATION
location_ID varchar(36)
address_line varchar(64)
city varchar(64)
county varchar(64)
country varchar(64)
COMPANY
company_ID varchar(36)
register_code varchar(64)
company_name varchar(100)
vat_number varchar(10)
email varchar(64)
phone varchar(64)
location_ID varchar(36)
ITEM_OWNERSHIP
item_ownership_ID varchar(36)
item_ID varchar(36)
app_user_ID varchar(36)
company_ID varchar(36)
INVOICE
invoice_ID varchar(36)
invoice_number varchar(20)
invoice_date date
vat_percent numeric(19, 0)
vat numeric(19, 0)
invoice_without_vat numeric(19, 0)
invoice_total numeric(19, 0)
app_user_ID varchar(36)
company_ID varchar(36)
APP_USER
app_user_ID varchar(36)
firstname varchar(64)
lastname varchar(64)
phone varchar(64)
location_ID varchar(36)
APP_USER_COMPANIES
app_user_companies_ID varchar(36)
company_ID varchar(36)
app_user_ID varchar(36)
PAYMENT
payment_ID varchar(36)
invoice_ID varchar(36)
app_user_ID varchar(36)
payment_type_ID varchar(36)
amount int4
payment_date date
company_ID varchar(36)
PAYMENT_TYPE
payment_type_ID varchar(36)
payment_type_description varchar(64)
Powered By Visual Paradigm Community Edition