Tallinna Tehnikaülikool

Online healthy food ordering

Home project

Marko Gordejev

185751IADB

Instructor: Andres Käver

Tallinn 2020

AUTHOR’S DECLARATION

I hereby certify that the work in question is the result of my personal work. All sources used to compile the work are cited. This work has never been published anywhere else before.

_____

PROJECT DESCRIPTION

The theme of this home project is - Online food ordering from a healthy restaurant.

We have a company called HealthyFoodAndMe that specializes in providing people with healthy food experiences by preparing food in a high-quality restaurant. Healthy foods and drinks can be ordered through the company's website. There is also the possibility to order different meals on the spot in the restaurant. The restaurant is represented in several towns around the country to ensure that a large number of people can have access to healthy and tasty food. In each town there are a certain amount of restaurants that have been divided across areas in the town.

Every HealthyFoodAndMe restaurant is staffed by a number of chefs as well as cleaners and other workers. The role and the working time of the restaurant worker is brought out in the Person_in_restaurant entity. Customers of the restaurant have the opportunity to choose from a wide selection of healthy foods and drinks they want to add to their order. After they have finished with adding items to their order, they are then presented a bill, which they have to pay for. Work on the order will begin by the restaurant's chefs after the customer has paid at least half the price for the order. The status of the order will be updated when it is advanced in the making. Customers also have the option to pay for the order in installments. For example, half the price of the order via the web and the other half when receiving the product. As a result, customers also have several options for paying their bill - cash (on the spot), by card or by wire transfer.

The restaurant offers people a variety of healthy, wholesome meals that are balanced and filled with macronutrients, vitamins and fresh ingredients. Customers also have the opportunity to make a food item of their choice by choosing the ingredients they like. All the ingredients, food and drink items will be added to the customers order. The customer can also choose the type of order - whether the food will be delivered to them or they will collect it themselves. Each food item also has its own type. For example, you have the option of choosing between salty meals and sweet food items.

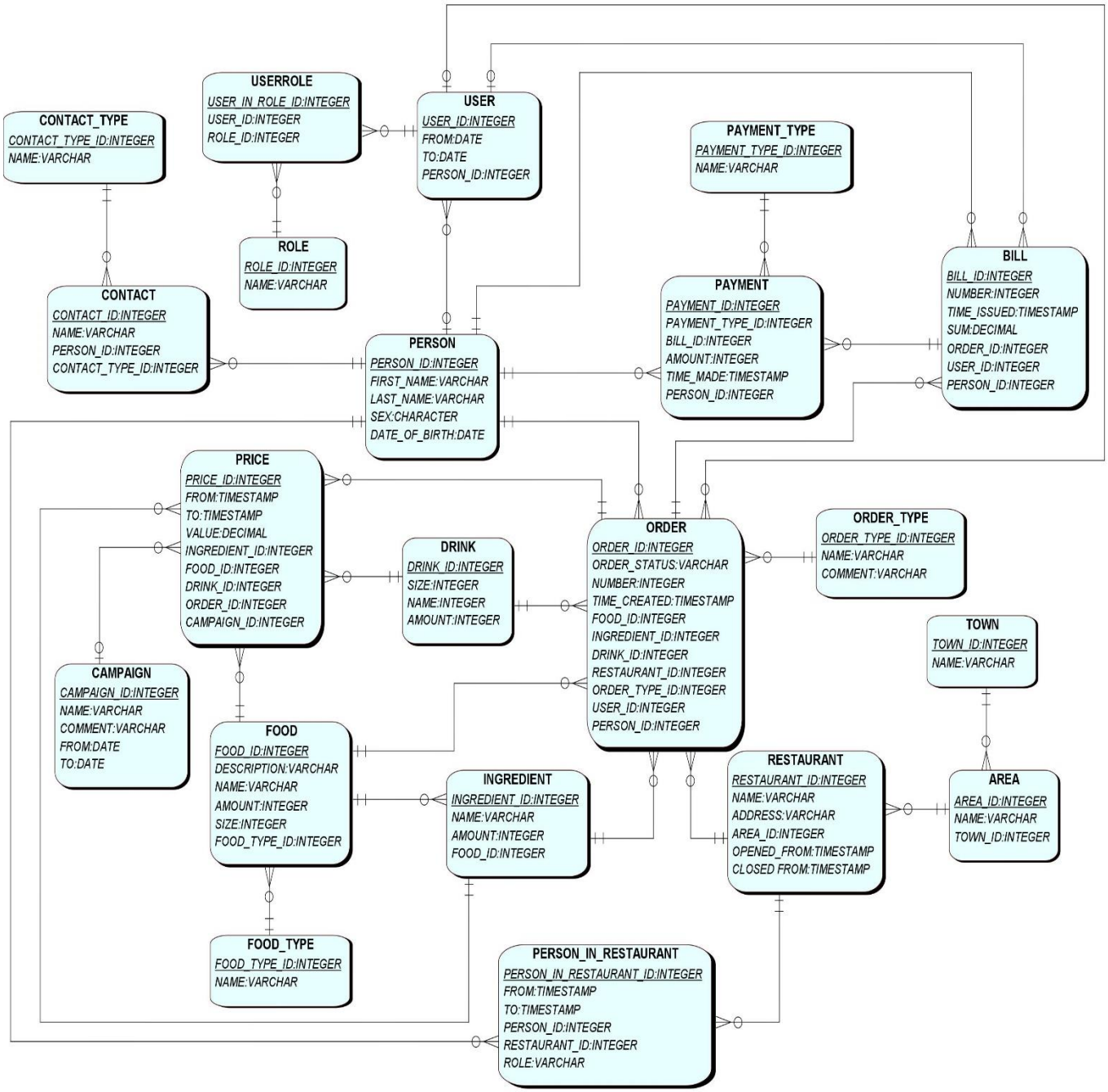
People have the opportunity to create a user on the restaurant's website, which makes the user experience much easier and more convenient. Registered users can see their previous orders and if they were satisfied with their previous order then they can re-place the order with a single click of a button. Naturally, it is not mandatory for the person to have an account on the restaurants website to make an order. If the person does not have an account and wants to order a product from the website then he/she must write down his/her contact details before submitting the order. The website also has an administrator who performs daily tasks that are required to maintain a website. The role of the administrator is brought out in the Role entity

From time to time, the restaurant also arranges campaigns on its products. When a campaign is announced on an item, the price will be discounted on the given item.

I got the motivation for my project idea as I had to prepare a business plan within the subject Ettevõtluse alused (ICY0029). After a long period of thought, the idea came to me to create a restaurant company that would serve people who value delicious and healthy foods. It would

be extremely important for the company to have a website, as this would make it easier for customers to order their favorite dishes and drinks.

ERD


USERROLE
USER_IN_ROLE_ID:INTEGER
USER_ID:INTEGER
ROLE_ID:INTEGER
USER
USER_ID:INTEGER
FROM:DATE
TO:DATE
PERSON_ID:INTEGER
CONTACT_TYPE
CONTACT_TYPE_ID:INTEGER
NAME:VARCHAR
PAYMENT_TYPE
PAYMENT_TYPE_ID:INTEGER
NAME:VARCHAR
ROLE
ROLE_ID:INTEGER
NAME:VARCHAR
BILL
BILL_ID:INTEGER
NUMBER:INTEGER
TIME_ISSUED:TIMESTAMP
SUM:DECIMAL
ORDER_ID:INTEGER
USER_ID:INTEGER
PERSON_ID:INTEGER
CONTACT
CONTACT_ID:INTEGER
NAME:VARCHAR
PERSON_ID:INTEGER
CONTACT_TYPE_ID:INTEGER
PERSON
PERSON_ID:INTEGER
FIRST_NAME:VARCHAR
LAST_NAME:VARCHAR
SEX:CHARACTER
DATE_OF_BIRTH:DATE
PAYMENT
PAYMENT_ID:INTEGER
PAYMENT_TYPE_ID:INTEGER
BILL_ID:INTEGER
AMOUNT:INTEGER
TIME_MADE:TIMESTAMP
PERSON_ID:INTEGER
PRICE
PRICE_ID:INTEGER
FROM:TIMESTAMP
TO:TIMESTAMP
VALUE:DECIMAL
INGREDIENT_ID:INTEGER
FOOD_ID:INTEGER
DRINK_ID:INTEGER
ORDER_ID:INTEGER
CAMPAIGN_ID:INTEGER
DRINK
DRINK_ID:INTEGER
SIZE:INTEGER
NAME:INTEGER
AMOUNT:INTEGER
ORDER
ORDER_ID:INTEGER
ORDER_STATUS:VARCHAR
NUMBER:INTEGER
TIME_CREATED:TIMESTAMP
FOOD_ID:INTEGER
INGREDIENT_ID:INTEGER
DRINK_ID:INTEGER
RESTAURANT_ID:INTEGER
ORDER_TYPE_ID:INTEGER
USER_ID:INTEGER
PERSON_ID:INTEGER
ORDER_TYPE
ORDER_TYPE_ID:INTEGER
NAME:VARCHAR
COMMENT:VARCHAR
TOWN
TOWN_ID:INTEGER
NAME:VARCHAR
CAMPAIGN
CAMPAIGN_ID:INTEGER
NAME:VARCHAR
COMMENT:VARCHAR
FROM:DATE
TO:DATE
FOOD
FOOD_ID:INTEGER
DESCRIPTION:VARCHAR
NAME:VARCHAR
AMOUNT:INTEGER
SIZE:INTEGER
FOOD_TYPE_ID:INTEGER
INGREDIENT
INGREDIENT_ID:INTEGER
NAME:VARCHAR
AMOUNT:INTEGER
FOOD_ID:INTEGER
RESTAURANT
RESTAURANT_ID:INTEGER
NAME:VARCHAR
ADDRESS:VARCHAR
AREA_ID:INTEGER
OPENED_FROM:TIMESTAMP
CLOSED FROM:TIMESTAMP
AREA
AREA_ID:INTEGER
NAME:VARCHAR
TOWN_ID:INTEGER
FOOD_TYPE
FOOD_TYPE_ID:INTEGER
NAME:VARCHAR
PERSON_IN_RESTAURANT
PERSON_IN_RESTAURANT_ID:INTEGER
FROM:TIMESTAMP
TO:TIMESTAMP
PERSON_ID:INTEGER
RESTAURANT_ID:INTEGER
ROLE:VARCHAR

ANALYSIS OF SOFT DELETE/UPDATE

I was given a task to implement soft delete and soft update in single tables, 1:m (one-to-many relationship) and 1:0-1 (one-to-one optional relationship) tables with SQL sentences. The program where I wrote my SQL sentences was Azure Data Studio. The MS SQL Server was provided by the instructor.

The main advantage of soft delete over an actual physical delete of a record in a database is that, when soft deleting data, it is not physically deleted, but rather discarded so that data can easily be recovered when something gets deleted by accident or when the record history of a table needs to be checked upon on.

Before starting with the task, I searched the internet for information about soft delete/update and I gathered, that there are several ways to implement soft delete and soft update in databases: First of all there is the way of having a boolean attribute in an entity that simply states whether a record has been soft deleted or not. I also found a way of creating a table that would mirror the structure of the table I would like to soft delete a record from, and when soft deleting I would simply put the record into the deleted data table (I didn't choose this solution because keeping the data structures in sync would be too complicated). Then there is the possibility of having a datetime field for marking the time the record was soft deleted. Since in the given task it was required for the database to support time change, I decided to implement the datetime choice. The method to implement soft delete/update with the datetime attribute that I found was the most understandable (after much investegation and research) to me, was with composite keys i.e combinations of two or more columns in a table that together uniquely identify an entity occurrence. For the composite key values I decided on the ID and the DeletedAt attributes of the record.

In composite primary keys it is not allowed to have a nullable field, as null cannot be compared to any value, as it does not have a known value (i.e it's value is undefined). In my entities I decided to create a composite key of the table's primary key and the datetime attribute which indicates when a record was deleted (DeletedAt). As the composite key cannot have a null value in it, I decided to put a placeholder datetime into the DeletedAt field of 9999-12-31 as this is the maximum data range value of DATETIME2. The reason for setting it to the maximum value lies in the fact that if a record has not been deleted, it's DeletedAt time will be in the long future.

I also derived, that having a foreign key as a nullable means that it is optional i.e that the connection is a optional one-to-many relationship, which was not required for this given task.

I created a database that had 4 tables – PERSON and CONTACT, which were related with a one to many relationship and also STUDENT and ADDRESS, which were related with a one-to-one optional relationship. Then I inserted some data into the tables and began experimenting with SQL sentences.

REPOSITORY PATTERN ANALYSIS

Repository pattern is an abstraction of the data access layer. It is essentially a way to implement data access by encapsulating the set of objects persisted in a data store. It is one of the most popular patterns to create an enterprise level application, or any kind of application for that matter. It is used mainly to decouple the business logic and the data access layers in the application.

The point of repository pattern, is to make code easier to read, test and to extend further. It also removes unwanted duplication that makes your code longer and harder to read. It also gives the possibility to replace certain parts of the project without the need to change the API. The repository pattern works by seperating the projects data access logic from the rest of the code and by mapping it to the entities that are located in the domain class.

There are a handful of ways of how to implement the repository pattern. For an example, it is possible to make a generic repository interface which would ensure that you have a common interface for working with any of the objects. This option would also grant the possibility to implement the interface generically as well. This would in turn, give you the option to easily create repositories without having to write any new code. Another way would be to create a repository of each separate business object. Each possibility has its own pros and cons. The choice which one you should implement lies within yourself or however your project team decides to do it.

Data Access Object (DAO) in essence, is a pattern to persist domain objects into a database. DAO and the repository pattern are both ways of how to implement DAL. The point of DAL is to keep the code clean by isolating the database access logic. Comparing the repository pattern with the Data Access Object pattern one can see few minor differences. First of all DAO is an abstraction of data persistence, whereas a repository is an abstraction of a collection of objects. Second of all, DAO is more table-centric, whereas a repository deals only in aggrergate roots. In summary, DAO pattern offers only a loosely defined contract, which in turn makes it suffer from getting bloated implementations. The repository pattern however, uses a metaphor of a collection and this in turn gives the pattern a tight contract, making it easier to understand.

In my project I have implemented a repository of each business object. I have each objects repository in the repository folder and there each repository derives from the entity framework base repository class which in turn derives from the base repository. At the same time each business objects repository derives from the same object repositorys name interface and this gives the possibility to add individual custom methods to any given repositories.