Project equivalent to written exam

Dragoş-Alexandru Efrim
Olivia Drobot

May 2023

1 Problem statement

Starting from a database that is already in NF3, create a program that allows the user to perform CRUD (Create, Read, Update, and Delete) operations on the database. For this exercise, JPA annotations can be used, and to connect the database to IntelliJ, PostgreSQL will be used. Finally, write the steps for running the application for anyone who would like to test the project.

2 Documentation

2.1 Links

- JetBrains - Used to create the application
- GeeksForGeeks - Used to connect to the PostgreSQL database
- BezKoder - Used to create models, repositories, and controllers

3 Main programming tools used

- We configured the Spring Boot project to set the dependencies and have access to Jakarta Persistence, as well as other utilities, such as:
  - @CrossOrigin - allows cross-origin resource sharing (CORS), enabling requests from a different domain to access the API endpoints in the controller
  - @RestController - marks the class as a RESTful controller; it combines the @Controller and @ResponseBody annotations, indicating that the class will handle HTTP requests and return the response directly as the body of the response
  - @RequestMapping - specifies the base URL mapping for all the endpoints in the controller
  - @Autowired - used for automatic dependency injection in Spring

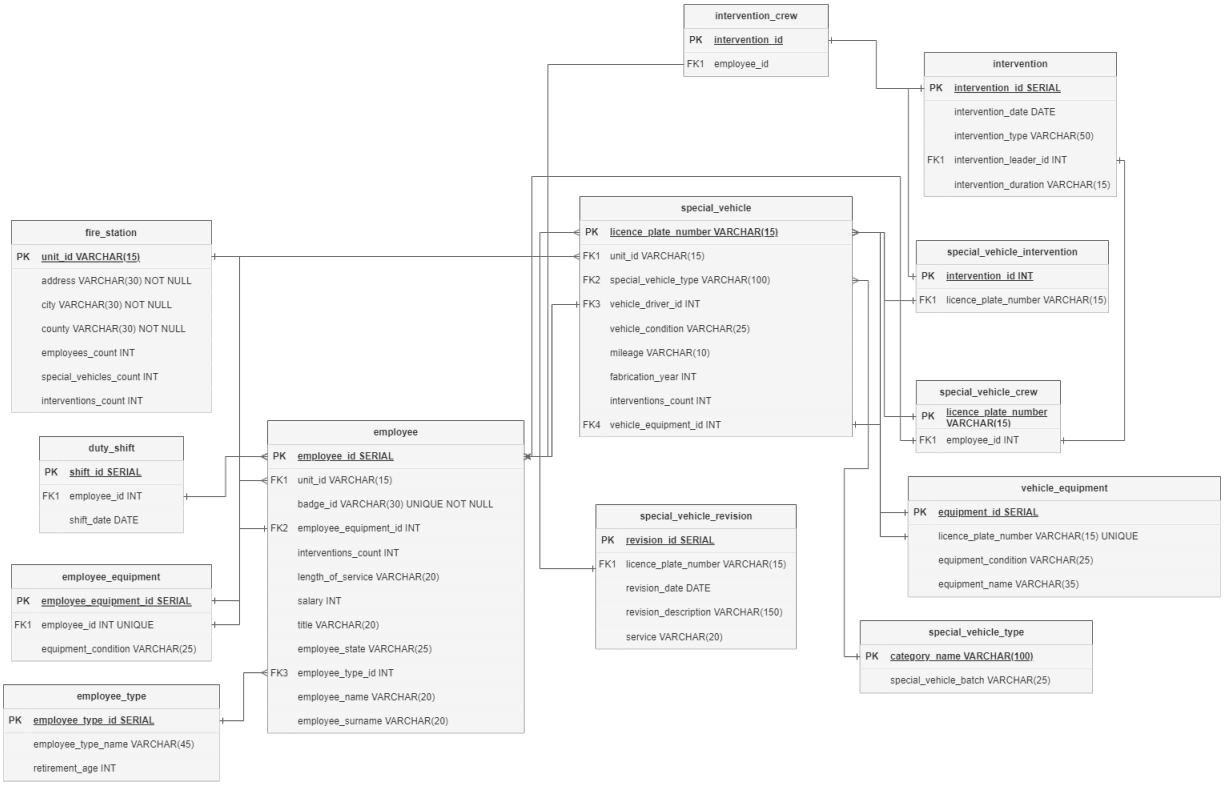
  - @GetMapping - maps the HTTP GET request to the specified endpoint; it handles the retrieval of all objects from the database
  - @PostMapping - maps the HTTP POST request to the specified endpoint; it handles the creation of a new object in the database
  - @PutMapping - maps the HTTP PUT request to the specified endpoint; it handles the updating of a specific object identified by it's ID path variable
  - @DeleteMapping - maps the HTTP DELETE request to the specified endpoint; it handles the deletion of all objects (or a specific one) from the database
  - @RequestParam - is used in the method to specify a query parameter that is optional; it allows filtering based on the provided parameter

- We have used Jakarta Persistence API (JPA) for storing, accessing and administrating data between the objects from Java and the PostgreSQL database
  - @Entity - used to mark the class as an entity, representing a persistent object that is mapped to a database table; it indicates that instances of this class can be stored and retrieved from the database
  - @Table - specifies the name of the database table to which the entity is mapped
  - @Id - used to mark the field as the primary key of the entity; indicates that this field uniquely identifies each object
  - @GeneratedValue - specifies the strategy for generating the values of the primary key
  - @Column - maps the specific field to its corresponding column in the database table; it specifies the name of the column to which this attribute is persisted

For organizing the project we have used multiple models, repositories and controllers:

- Models - contains the classes that represent the data models of the application; these are Java objects that are mapped to database tables
- Repositories - contains the interfaces that handle data access and manipulation in the database
- Controllers - contains the classes that define the entry points of the application and handle the HTTP requests received from the client

4 EER



5 Project contribution

- Dragoş-Alexandru Efrim
  - created models, repositories and controllers
  - created Postman requests
  - recreated database (in English)
- Olivia Drobot
  - realised database connection in application.properties
  - added project tests
  - research upon information regarding JPA