Relational Database Project Report

Table of Contents

[Introduction 1](#_Toc22855940)

[Requirements 1](#_Toc22855941)

[Working Hypotheses 1](#_Toc22855942)

[Specification 2](#_Toc22855943)

[Actors and Use Cases 2](#_Toc22855944)

[Application Dataflow 3](#_Toc22855945)

[Software Architecture 4](#_Toc22855946)

[Class diagram 4](#_Toc22855947)

[Database Conceptual Design 5](#_Toc22855948)

[Entities Definition 5](#_Toc22855949)

[Entities Attributes 5](#_Toc22855950)

[E-R Diagram (with Generalizations) 7](#_Toc22855951)

[Generalizations Resolutions 8](#_Toc22855952)

[Database Operations 9](#_Toc22855953)

[Database Volume Table 10](#_Toc22855954)

[E-R Diagram (final) 11](#_Toc22855955)

[Database Logical Design 12](#_Toc22855956)

[Database Tables 12](#_Toc22855957)

[Database Normalization 13](#_Toc22855958)

[Database Schema 14](#_Toc22855959)

[Class Diagram (Final) 15](#_Toc22855960)

Introduction

The following paper illustrates the design steps and the choices made for the development of an application based on a relational database devised to meet the data management requirements of a fictional tech company named *Innovative Solution*.  
  
The first part of the paper covers the application’s functional and non-functional requirements from which the application’s specifications are derived, consisting among other things of the actors involved and their use cases, the application dataflow, and an overall description of the software architecture.

The second part of the paper focuses on the database conceptual design, with the identification of the entities of interest and their relationships, which after resolving any generalization lead to the final E‑R diagram.   
  
The third and final part of the paper will present the classes derived from the entities and relationships identified in the database conceptual design and their Java implementation, which through the Hibernate framework will be mapped into tables in the underlying MySQL database.

Requirements

*Innovative Solutions* is a company whose core business consists in the retailing of electronic IoT-oriented products assembled in-house by teams of employees from components purchased from various suppliers.

Functional Requirements

The company requires a software solution which should be used by its system administrator to oversee and control all the information concerning the company, by its team leaders, who should be able to add finished products to the company stock and review information on its team members, and by the company’s customers, who will be allowed to purchase the products and review their orders.

Non-Functional Requirements

Since the application should not require computer expertise to be used, especially by customers, to enhance the non-functional requirement of usability a graphical interface should be provided for the application.

In addition, due to the company’s system administrator strong background in object-oriented programming, the application should be developed attuning to an object-oriented paradigm, concealing as much as possible the data persistence aspects of the application.

Working Hypotheses

The design of the application will also rely on the following working hypotheses:

* *Customers* may purchase any *Products* up to their available quantities
* Each *Employee* in the company may belong to up to one *Team*
* Each *Team* is composed of at least one member, representing its leader
* Each type of *Product* offered by the company is assembled by a single team

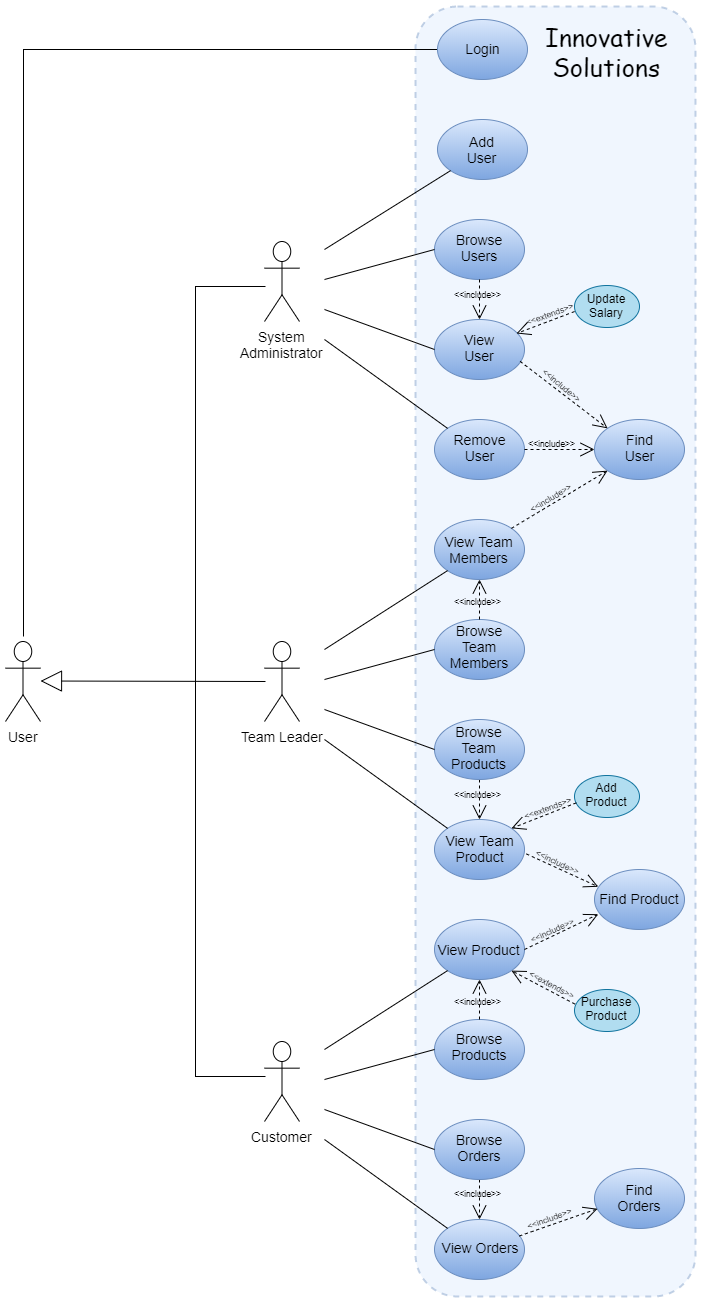
Specification

Actors and Use Cases Diagram

Given the software functional requirements, the application is meant to be used by three different actors, each being allowed to perform a different set of operations:

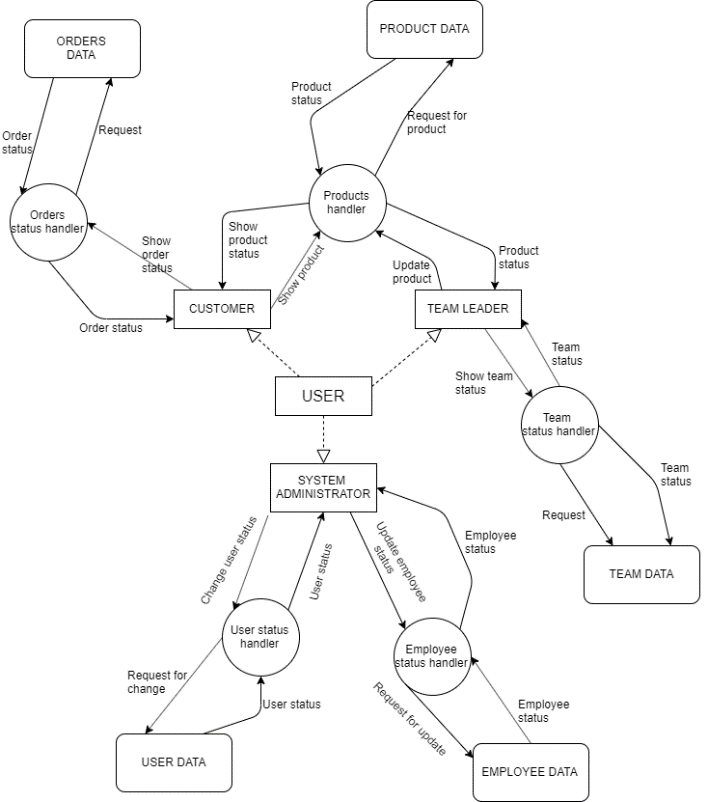
* The company’s system administrator, who is permitted to perform a number of actions requiring a high level of privilege, such as inserting a new customer or employee into the database or modify the salary of an employee.
* The team leaders, who are allowed to add finished products to the company stock and review information on the member assigned to their team.
* The customers, who may view and purchase the products on offer, as well as review their past orders.

Hence, the diagram of the application’s use cases diagram appears as follows:



Application Dataflow

The projected dataflow of the application is outlined below:



Software Architecture

The application will be written in the *Java* programming language, where the graphical interface will be developed through the JavaFX API, while the data persistence aspects of the application will be handled by the *Hibernate* framework implementation of the Java Persistence API, which by performing an Object-Relational Mapping (ORM) will transparently exchange the data between the application and an underlying MySQL database.  
  
More precisely, the software architecture can be divided into the following tiers:

[Immagine Software Architecture]

Database Conceptual Design

Here follows the conceptual design of the MySQL database of the application, where the entities and relationships identified will subsequently be mapped into classes to allow the management of the database through hibernate.

Entities Definition

From the application’s requirements and the additional working hypotheses the following entities are identified as the first step in the conceptual design of the database:

* **USER**: A generic user of the application, that is the system administrator, the team leaders or the company’s customers
* **CUSTOMER**: The company’s customers, who may purchase products and review their  
  orders
* **EMPLOYEE**: The company’s employees, where each employee may belong to a single team
* **TEAM:** The company’s teams, which are composed of a team leader and other members, where each team is assigned the assembly of one or more products.
* **PRODUCT**: The list of products offered by the company, which can be bought by customers
* **PRODUCT\_STOCK**: Represents the products the company has in stock, and so available for purchase by the customers

Entities Attributes

The list of the attributes related to each entity, including its identifiers, is outlined in the table below:

|  |  |
| --- | --- |
| ENTITY | ATTRIBUTES |
| USER | username, name, surname, password, mail |
| CUSTOMER | IDcostumer, address |
| EMPLOYEE | IDemployee, role, salary, team |
| TEAM | IDteam, location, teamLeader |
| PRODUCT | productName, productDescription, productPrice, productAvailability, team |
| PRODUCT\_STOCK | IDproduct, productName |

* An instance of the USER entity is uniquely identified by its username, which in conjunction with its password allows a user to access the application.  
  Other information related to a user consists in its real name and surname and its email.
* An instance of the CUSTOMER entity is uniquely identified by its IDcustomer, where the customer’s address is also recorded for product shipping purposes.
* An instance of the EMPLOYEE entity is uniquely identified by its IDemployee, and additional information related to an employee is given by his role in the company, its salary, and the team he belongs to, if any.
* An instance of the TEAM entity is uniquely identified by its IDteam, where the working location and the leader of each team is also recorded.
* An instance of the PRODUCT entity, representing a product offered by the company, is uniquely identified by its productName, where additional information on the product type such as a brief description, its current asking price, its current availability in stock and the team assigned to its assembly are also recorded.
* An instance of the PRODUCT\_STOCK entity, representing a single product in stock, is uniquely identified by its IDproduct, where the type of product is also recorded.

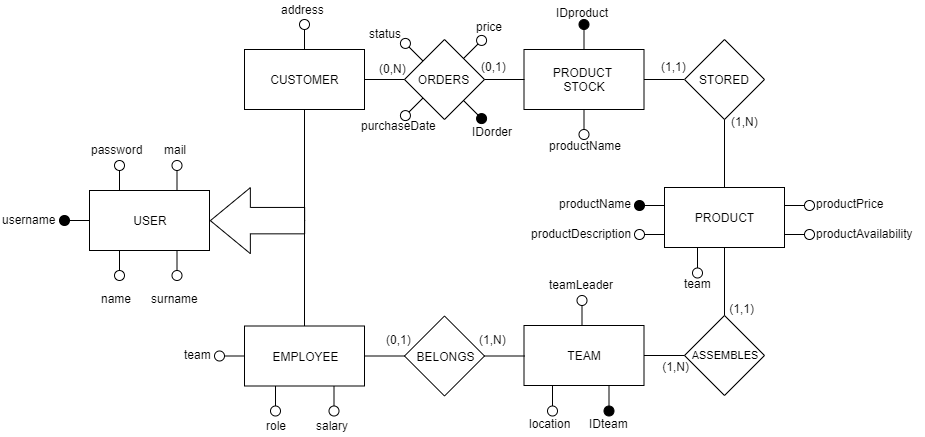
Entity Relationships

The following table describes the relationships between the entities identified in the previous step, along with their cardinality and description

|  |  |  |  |
| --- | --- | --- | --- |
| RELATION | ENTITY A (cardinality) | ENTITY B (cardinality) | DESCRIPTION |
| BELONGS | EMPLOYEE (0,1) | TEAM (1,N) | An Employee may belong to up to one team, while a team is composed of at least one member (its team leader) |
| ASSEMBLES | TEAM (1,N) | PRODUCT (1,1) | A Team assembles one or more different types of products, while each type of product is assembled by a single team |
| ORDERS | CUSTOMER (0,N) | PRODUCT\_STOCK (0,1) | A Customer may purchase zero or more products in stock, while each single Product in stock may be sold to a single Customer |
| STORED | PRODUCT\_STOCK (1,1) | PRODUCT (1,N) | Each Product in stock belongs to a single product type, while for each product type there may be one or more products in stock |

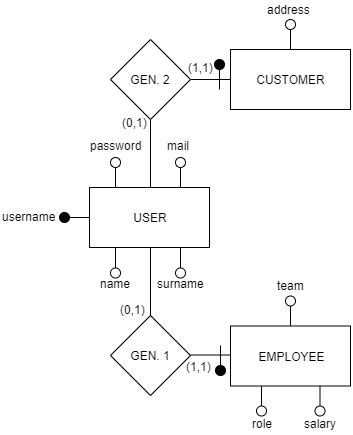
E-R Diagram (with Generalizations)

As for the entities and their relationships identified in the first part of the database conceptual design, the database E-R Diagram appears as follows:



Generalizations Resolutions

From the E-R diagram the USER entity represents a generalization of the CUSTOMER and EMPLOYEE entities, and our design choice was to resolve this generalization by not aggregating the father entity (USER) into the two children entities (CUSTOMER, EMPLOYEE).  
This solution, though involving an increase in the average number of accesses to the database, makes it possible to avoid the use in the USER table of NULL values for the attributes relative to the employees for the customers and for the attributes relative to the customers for the employees  
Thus the resolved generalization appears as follows:



Database Operations

In compliance with the application’s requirements and the use cases we have defined, the following operations on the database are taken into account:

|  |  |  |  |
| --- | --- | --- | --- |
| OPERATION | ACTOR | DESCRIPTION | EXPECTED FREQUENCY (times/day) |
| ADD USER | Administrator | Allows the administrator to add a User, whether an Employee or a Customer, to the database | 3 |
| REMOVE USER | Administrator | Allows the administrator to remove a User from the database | 1 |
| UPDATE SALARY | Administrator | Allows the administrator to modify the salary of an Employee | 1 |
| SHOW TEAM PRODUCTS | Team Leaders | Allows a team leader to view the list of Products assembled by his Team | 200 |
| SHOW TEAM MEMBERS | Team Leaders | Allows a team leader to view the members of his Team | 6 |
| ADD PRODUCTS | Team Leaders | Allows a team leader to add one or more finished products to the stock | 60 |
| SHOW PRODUCTS | Customers | Allows a Customer to view the list of products currently available for purchase | 200 |
| ADD ORDER | Customers | Allows a Customer to purchase an available Product | 20 |
| SHOW ORDERS | Customers | Allows a Customer to review his past orders | 100 |

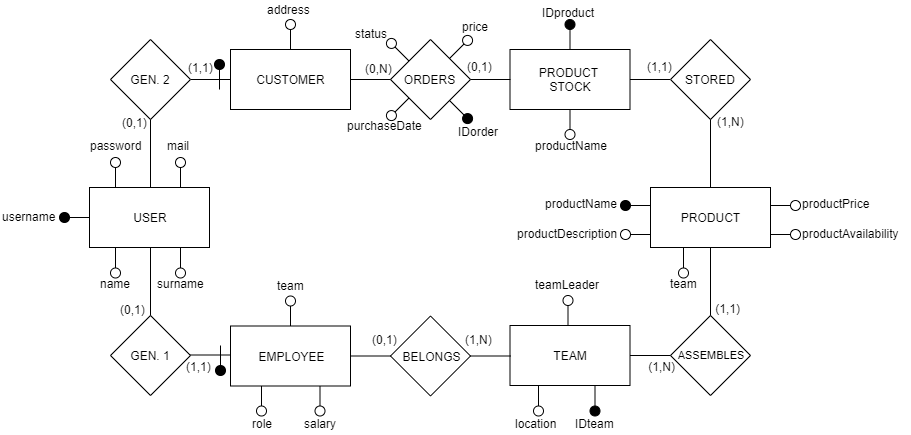
Database Volume Table

The following table shows the expected number of instances of the entities and relationships previously identified in the E-R diagram based on a projected usage of the application:

|  |  |  |  |
| --- | --- | --- | --- |
| NAME | E/R | EXPECTED INSTANCES | MOTIVATION |
| USER | E | 100 | Initial Hypothesis |
| GEN.1 | R | 30 | We assume 30% of the users to be Employees |
| EMPLOYEE | E | 30 | (1,1) cardinality with GEN.1 |
| GEN.2 | R | 70 | We assume 70% of the users to be Customers |
| CUSTOMER | E | 70 | (1,1) cardinality with GEN.2 |
| ORDERS | R | 350 | We assume each customer to place 5 orders on average |
| BELONGS | R | 27 | We assume 90% of the Employees to belong to a Team |
| TEAM | E | 6 | We assume the presence of 6 teams in the company |
| ASSEMBLES | R | 30 | We assume that each team assembles 5 products on average |
| PRODUCT | E | 20 | We assume the company to offer 20 different types of products |
| STORED | R | 600 | We assume the company to keep an average stock of 30 instances for each product type |
| PRODUCT\_STOCK | E | 600 | (1,1) cardinality with STORED |

E-R Diagram (final)

The final E-R Diagram result of the database conceptual design is outlined below:



Hibernate Database Implementation

Classes Diagram

From the entities and the relationships identified in the final E-R diagram we can derive the following classes, which once implemented in Java will allow for a transparent database management through the use of Hibernate.

[Insert Class Diagram Here]

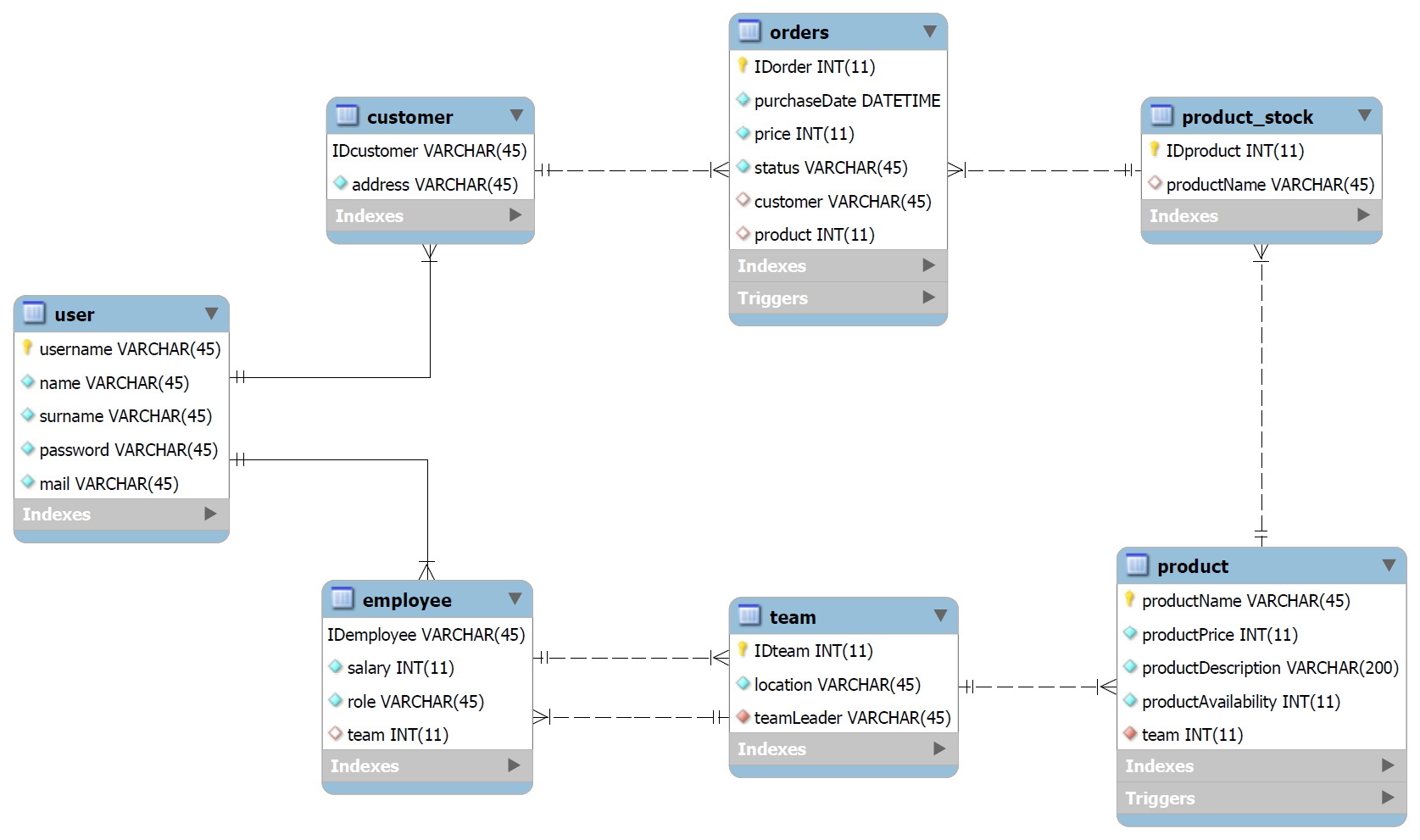
Classes Implementations

Here we present the Java implementation of the classes, with the annotations required by Hibernate to create and manage the database already present.

Persistence.xml

Hibernate Database Schema

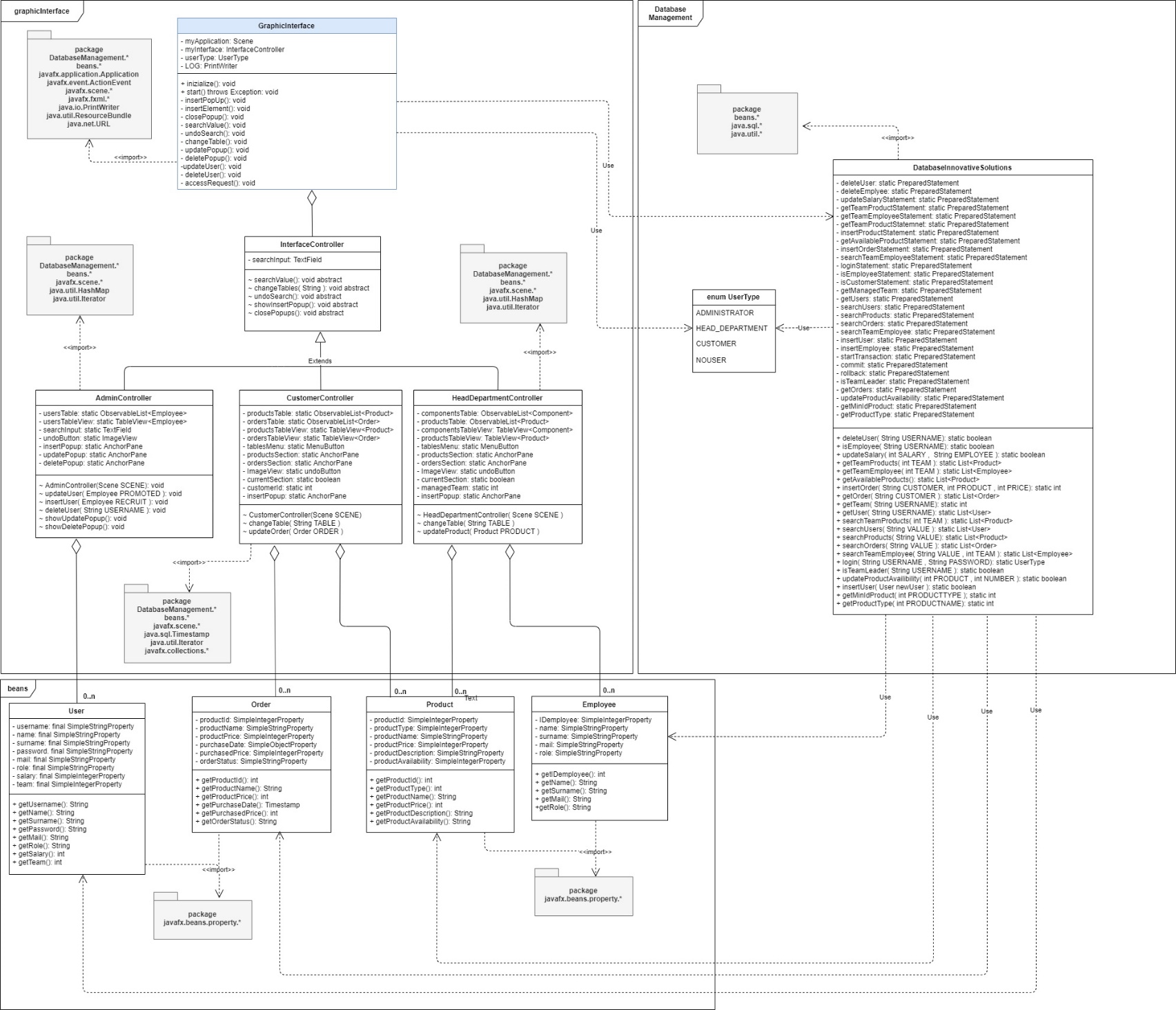
The SQL database schema generated by Hibernate from the classes we have defined is shown below



**AGGIORNARE**

Software Architecture Diagram (Final)

The final UML diagram of the software architecture is shown below:



**AGGIORNARE**