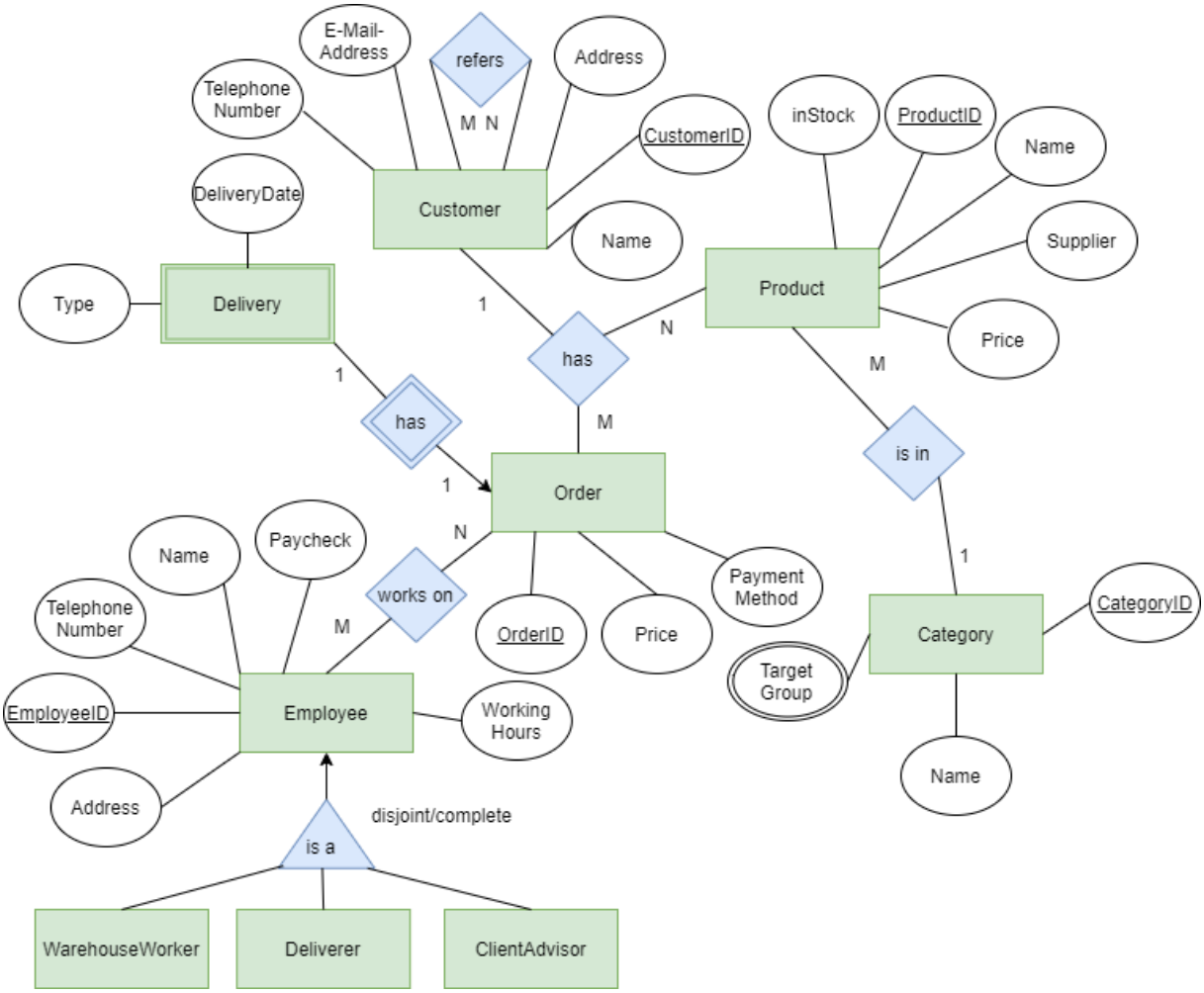
DATABASE SYSTEMS PROJECT - ONLINE SHOP
BY FLORIAN SCHWARZ 01505559

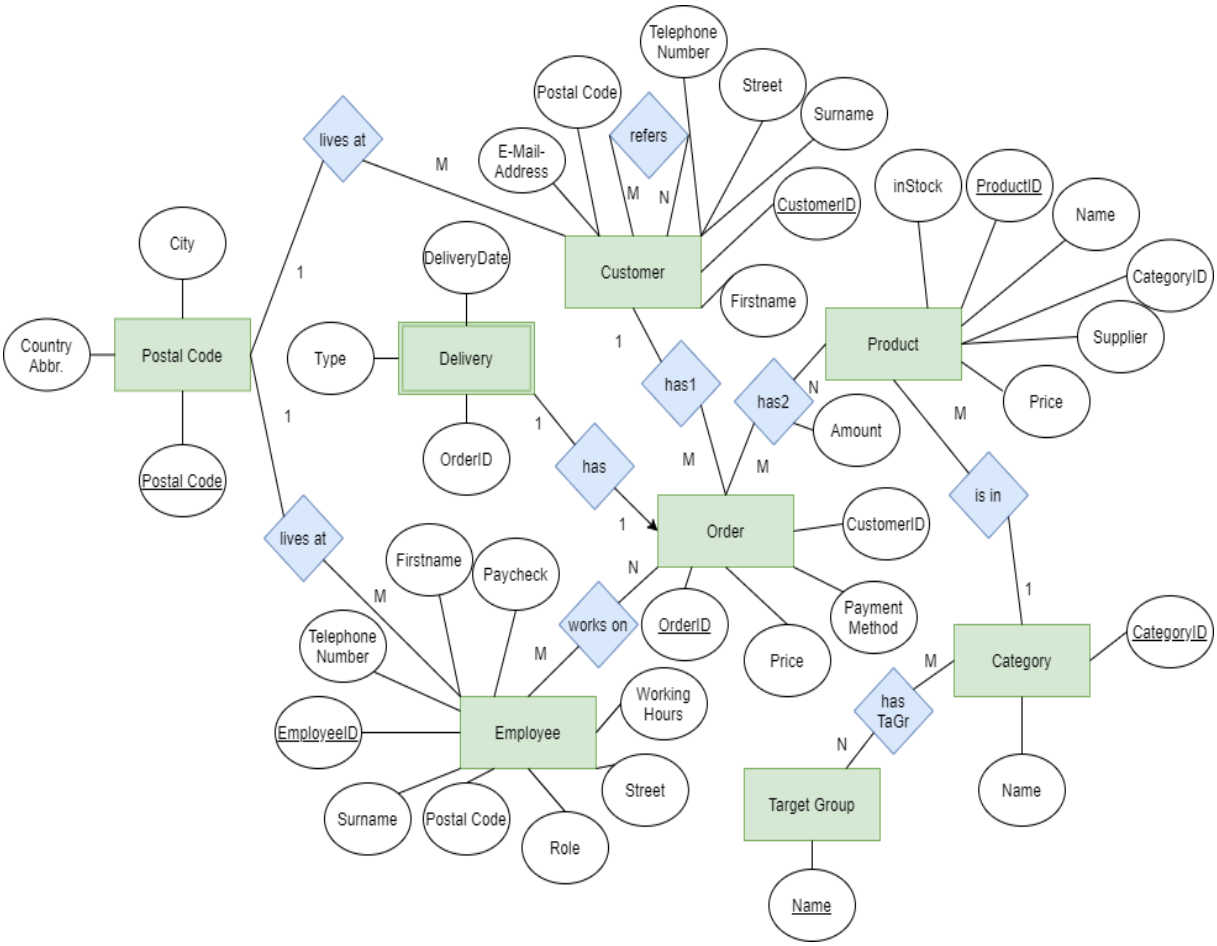
Milestone 1a: Requirements Analysis

An **ONLINE-SHOP** is centered around **Orders**, which are worked on by **Employees**. **Employees** have an *EmployeeID*, a *Firstname*, *a Surname*, a *Telephone Number*, *a Street Name* and live at a City with a certain *Postal Code,* get a certain *Paycheck* and work for a certain amount of *Hours*. The **Postal Codes** themselves are linked with *Cities* and *Countries*. Employees also are a certain Type of Employee, i.e. Warehouse Worker, Deliverer or Client Advisor, nothing else. They can not operate more than one role. **Orders** have a *Price*, a *Payment Method* and an *OrderID*. The *Price* is a sum of all *Prices* of **Products** contained in the **Order**. The *Payment Method* is "cash" by default. Every **Order** has exactly one **Delivery** that goes with it. This **Delivery** has a *Delivery Date* and a certain *Type*. **Orders** can be placed by **Customers** and contain certain **Products,** the amount of which is always noted. A **Customer** has a *Firstname, a Surname,* lives at a City with a certain Postal Code, an *Email-Address* and a *CustomerID*. A **Customer** can refer other **Customers** to the Online-Shop, once per account (Meaning no Customer can refer the same Customer twice), in order to obtain a coupon. A **Customer** can't refer themself. A **Product** has a *ProductID*, exists in a certain amount in *Stock*, has a *Name*, a *Supplier* and a *Price*. Every **Product** is in precisely one **Category**, which itself has a *Name*, a *CategoryID* and certain **Target Groups** which are of importance for market research. A **Target Group** can also be interested in mutliple **Categories**. **Target Groups** themselves have a *Name* which identifies them.

Milestone 1b: Conceptual Design



Original ERD diagram satisfying all side constraints of MS1.



The final look of the ERD diagram of the domain, already with all tables in 3NF.

Milestone 2: Logical Design

ENTITIES

Order (OrderID, Price, PaymentMethod, *CustomerID*)
PK: OrderID
FK: OrderID.CustomerID ◇ Customer.CustomerID

Delivery (*OrderID,* Type, DeliveryDate)
PK: OrderID
FK: Delivery.OrderID ◇ Order.OrderID

Customer (EmailAddress, *postal_code*, street, CustomerID, firstname, surname, Telephone Number)
PK: CustomerID
FK: Customer.postal_code ◇ Postal_code.postal_code
Alternate Candidate Key: EmailAddress, Telephone Number

Product (ProductID, Price*, CategoryID*, Name, Supplier, inStock)
PK: ProductID

Category (CategoryID, Name)
PK: CategoryID

Employee (EmployeeID, firstname, surname, *postal_code*, street, Telephone Number, Paycheck, WorkingHours, Role):
Alternate Charakter Key: Telephone Number
FK: Employee.postal_code ◇ Postal_code.postal_code
PK: Employee ID

Postal_Code(postal_code, city, country_abbr)
PK: postal_code

TargetGroup(Name):
PK: Name

RELATIONS

worksOn (*OrderID*, *EmployeeID*)
PK: OrderID, CategoryID
FK: worksOn.OrderID ◇ Order.OrderID
worksOn.CategoryID ◇ Category.CategoryID

refers (*CustomerID1, CustomerID2*)
PK: CustomerID1, CustomerID2
FK: refers.CustomerID1 ◇ Customer.CustomerID
refers.CustomerID2 ◇ Customer.CustomerID

has1 (*ProductID, CustomerID*)
PK: ProductID
FK: has.ProductID ◇ Product.ID
has1.CustomerID ◇ CustomerID

has2 (*ProductID, OrderID*)
PK: ProductID, OrderID
FK: has.ProductID ◇ Product.ID
has.OrderID ◇ Order.ID

hasTargetGroup(*TGName,CategoryID*)
PK: TGName, CategoryID
FK: TGName, CategoryID
hasTargetGroup.TGName ◇ TargetGroup.Name
has.CategoryID ◇ Category.CategoryID

Milestone 4: Implementation

Normalization:

Compare with the initial ERD in order to see the initial situation before the normalization process.
The first problem was encountered in the Employee Table, since originally the address contained the country, postal code, city and street adress. However, the city is dependant on the postal code, which would lead to redundancy within the table. Therefore, we need to introduce a new table, namely Postal_Codes, which stores postal codes and links them with cities as well as the country that they are in.

PostalCode -> CountryAbbr
PostalCode -> City

Note how PostalCode is not a Superkey for the original Table Employee (Since EID is needed as well) and the respective right sides are WEAK attributes. Therefore this table was not originally in 3NF.
The exact same principle was used in the table of Customer, it also references the Postal Code table now (see new ERD).

Additionally, the terniary relationship had a problem. It originally looked the following way:

Has(OrderID,ProductID,CustomerID):
OrderID -> CustomerID
ProductID -> ProductID

Again, the left sides are not Superkey and the right sides are non prime attributes, therefore 3NF does not hold. Instead, two new tables are created in has1 and has2 (refer to MS 2), which now are both 3NF-conform.

TestData Insertion:

The files “javaconform_<tablename>.txt” were created externally (using python), so that they could be read by csv reader. Within Java, the insertion of this data was implemented, as well as the foreign key handeling. For instance, when inserting into the has2 table, Java chooses random Products out of the database. Since in has2, the primary key consists of (ProductID,OrderID) using the same Product in the same Order twice has to be avoided. This is achieved by storing already used ProductIDs in an array and checking whether a newly chosen ProductID was already used (compare with insertIntohas2() in DatabaseHelper.java). A similar tactic is applied when assigning employees to orders in insertIntoworksOn(). Please, refer to the source code file for further explanation of the code.
In order to produce citynames, firstnames, surnames, telephone numbers, ect.. online generators were used. The city postal codes and countries associated do not reflect reality.
When necessary, information was inserted by hand, for instance Product Names, to create an authentical look of the database.
Generally, the originally provided example functions for insertions were used and rebuilt accordingly.

Application:

The Application of the Domain of an Online-Shop spans 6 relations in total, namely:
Postal_Code, Customer, Product, Order, Delivery and has2.
Therefore, the tables Employee, worksOn, Category, refers and TargetGroup were not implemented.

The php-application lets the user modify data in all these tables, but since has2 is dependent on the tables Product and Order, one may not directly modify has2. This is instead done in the background, when modifying the table Order.
Insertion is possible in all tables but Delivery, since it is tied with Order. Updating is possible in all tables except Postal_Code and Order, since for these tables an update feature would not make much sense. Deletion is possible in all tables except Delivery, for the same reason as Insertion. Searching is possible in all tables.
Stored procedures have been used in order to implement the Deletions, therefore are very simplistic, however return an Error and subsequently tell the user whether the deletion was successful or not.
Since the attributes of Price in Order and inStock in Product are dependant on insertions in other tables, they are updated using php. For instance, when creating a new Order, using a certain Product with a specific amount, this amount is getting deducted from the InStock attribute of the respective entry in Product.
Bootstrap was used in a simplistic manner in order to polish the look of the website.