# Programming Assignment 2 Report

|  |  |
| --- | --- |
| Student: | Christoffer Eid – ce223af[@student.lnu.se](mailto:ilir.jusufi@lnu.se) |

## Project Idea

Here you should discuss your idea. For instance:

For this assignment we have designed and implemented a system to store data about the COVID vaccination. We have downloaded the data from this source: [www.somesource.com](http://www.somesource.com/) (alternatively you can generate your own data). Our tool enables users to track this data but also provide various statistics and visualizations of the data. ….

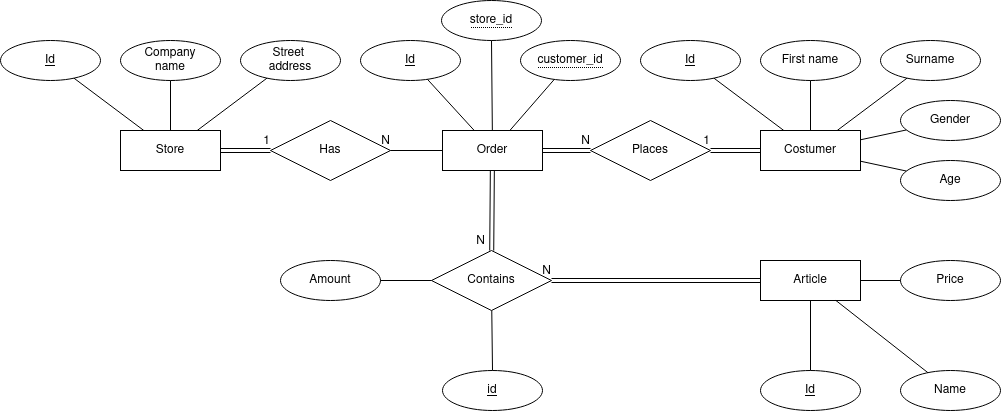
For this assignment I have designed and implemented a system to store customer, store, order and article data in one database.

I generated all the data using the tools supplied at [https://www.mockaroo.com](https://www.mockaroo.com/).

My program enable supervisor at a store chain to keep track of all its customers data and purchases, the chains supplied articles, all the chains stores different locations and more.

## Schema Design

Here you present your schema design. You can use softwares such as [https://app.diagrams.net](https://app.diagrams.net/) to draw your schema. Explain all the tables/relations and different connections they have.



All of my entities has an AUTO\_INCREMENTED id as primary key. This makes it easy to Query for the exact result the manager of the store wants to find. *Order* and *Article* has a many-to-many relationship. This makes it so that in the actual databse design, i had to create a relation table; *OrdersArticles* to keep track of what articles and how many of it there was in each order. The *OrdersArticles* table is represented by the relationship *Contains* inbetween the entities *Order* and *Article*.

Apart from this the other stuff is quite straight forward. There’s one customer per order, but a customer can have several orders, but atleast one. Because if a customer doesnt have an order, it isn’t a customer.

For each order there’s a store. And for each store, there can be any amount of orders. Here we even allow for a store to have 0 orders, since that will be the case any time we open a new store.

The *Order* entity keeps track of its connected store and customer by keeping their respctive primary keys as foreign keys.

## SQL Queries

V: **Combining *Orders*, *OrdersArticles* and *Articles* into one view.**

The following view is made up of a multirelation query and uses two *INNER JOINs.* We join table *Orders* on table *OrdersArticles* by matching the primary key *Articles.articleid* to the foreign key *OrdersArticles.article*. We then join that with table *Articles* by matching primary key *Articles.articleid* with foreign key *OrdersArticles.article.*

This view becomes really useful when querying from specific orders or shopping carts.

CREATE VIEW ordersandarticles AS

SELECT \*

FROM Orders

INNER JOIN OrdersArticles ON Orders.orderid=OrdersArticles.order

INNER JOIN Articles ON OrdersArticles.article=Articles.articleid;

Q: **List the average age of the customers in each store.**

The following query is a multirelation query and uses two *INNER JOINs,* just like our view. We do as we did with the view except on differnet tables. We also get to use our view here. Without our view this quey would have been double the length and four times as complex. What’s also interesting is this query is that we get to use aggregation. Whenever we do aggregation in a query, it alwasy has to be followed by a *GROUP BY*. In this query we wanted to know the average age of the customers in each store, hence we decide to *GROUP BY Stores.name.*

SELECT AVG(Customers.age), Stores.name

FROM Customers

INNER JOIN ordersandarticles ON customerid=ordersandarticles.customer

INNER JOIN Stores ON ordersandarticles.store=Stores.storeid

GROUP BY Stores.name

ORDER BY AVG(Customers.age);

Here you present and discuss the most interesting queries. Make sure you have 5 of them at least and check the specification in the assignment sheet. One example is found below:

Q: **List the name, last name and job title of the employee from a given city.**

The following query is a multirelation query and uses *JOIN.* We pass the argument of the city name (marked with ? in the query) and the query should give us all the employees of the corresponding shop. We join table *Employees* on table *Shops* by matching the *Shops.ID* to the foreign key *Employees.shopID*

SELECT firstname, lastname, jobtitle

FROM Employees

JOIN Shops ON Employees.shopID = Shops.ID

WHERE city=?;

## Discussion and Resources

Here you can write anything you might think it is important and provide the link to the required resources. For example:

We had issues with the missing and inconsistent data. We decided to remove/insert NULLS in the missing/corrupted attributes/tuples…...

The project uses xyz library, please check readme.txt for installation details.

Source code: [github/... link]

Video demonstration: [youtube/vimeo/... link]