

Pham Quang Tien,

Ly Duong Hai,

Le Thanh Thi

**La Tee Box Database Management Project**

Helsinki Metropolia University of Applied Sciences

Bachelor Degree

Information Technology

La Tee Box Database Management Project

5 December 2016

|  |  |  |
| --- | --- | --- |
| Author(s)  Title  Number of Pages  Date | Pham Quang Tien, Ly Duong Hai, Le Thanh Thi  project  28 pages  6 Dec 2016 | |
| Degree | Bachelor | |
| Degree Programme | IT | |
| Specialisation option | None. | |
| Instructor(s) | Aarne Klemetti, Senior Lecturer | |
|  | | |
| Keywords | |  |

Contents

[1 Introduction 2](#_Toc468828449)

[1.1 Background of the project topic 2](#_Toc468828450)

[1.2 Commissioning company introduction 3](#_Toc468828451)

[1.3 Steps of implementing the project 5](#_Toc468828452)

[2 User stories 6](#_Toc468828453)

[3 ER model 7](#_Toc468828454)

[3.1 Entities 8](#_Toc468828455)

[3.2 Relations between entities 10](#_Toc468828456)

[3.3 Normalization 10](#_Toc468828457)

[3.4 Specifying data type of the attributes 10](#_Toc468828458)

[4 DDL/SQL of the database 12](#_Toc468828459)

[4.1 CREATE 12](#_Toc468828460)

[4.2 ALTER 13](#_Toc468828461)

[4.3 DROP 15](#_Toc468828462)

[5 DML/SQL of CRUD + SELECT examples 16](#_Toc468828463)

[5.1 INSERT 16](#_Toc468828464)

[5.2 UPDATE 18](#_Toc468828465)

[5.3 DELETE 20](#_Toc468828466)

[5.4 SELECT examples 21](#_Toc468828467)

[6 Conclusion 25](#_Toc468828468)

[Electronic references 26](#_Toc468828469)

# Introduction

The project aims to create a working database for La Tee Box, a tea and coffee delivery startup company in Helsinki. The database is needed to enable as a tool to keep track on the information about customers, orders and inventory of the company. At the moment, the company is still using Excel files in order to manage the information, but with the company’s fast pace of growing, a more proper database can be useful to manage and access the data in a better manner.

La Tee Box offers an online payment method for customers in order to enhance the convenience of the service as well as monthly newsletters for interested potential customers or royal customers. The information will be then saved as entries and sent to the owners of La Tee Box and that information is then copied to an Excel file.

The mentioned process is not efficient, considering that any change of data would require the users to adjust many different tables. For example, if a subscriber cancels subscription, then his or her entry in the Customer table will need to be removed, along with the entry in Order table so that the number of orders remains correct. Therefore, the company needs a more advanced database system.

## Background of the project topic

The project is part of Data Management course in Metropolia University of Applied Sciences. The goal of the project is to create a working database that fulfil certain requirements. The group formed the idea of the project topic because a group member was working for the startup company La Tee Box.

While working at La Tee Box, he knew that the company needed a database to manage business data. With the knowledge of relational database obtained in Data Management course, he soon realized that a relational database would be much more convenient than working with Excel files. The company is still using Excel files in order to manage the information, but with the fast pace of growing of the company, a database can be useful to manage and access the data in a better manner.

The team members organized a meeting with the owners of La Tee Box to understand about their needs and expectations. The meeting pointed out that a database system can be very useful and reduce the amount of work for them. It also revealed that La Tee Box wanted simple application that could store different types of data and a way to sort and select needed data. The project team wanted to implement it based on the teacher’s instruction, which turned out to be quite close to the expectations of La Tee Box.

Based on the user stories, the goal of the project is then becoming quite clear. The company wanted a simple application, therefore the project team decided that **PhpMyAdmin** would be used, given that everyone in the company will be able to access the data. Therefore, the authors’ goal is to create a complete working database which is easy to use and manage.

## Commissioning company introduction

La Tee Box is a startup tea delivery service, operating in Helsinki and Espoo regions in Finland. The cusomter subscribe to the company’s tea plan for €19,95/month and get any type of tea they want. Customers can cancel subscription anytime free of charge. La Tee Box will ship out their La Tee Box on the 3rd of each month, so customers can expect the box to arrive a few days after. In each box, La Tee Box include two high quality and premium tea, 50 grams each. Moreover, so that customers can comfortably enjoy our unique loose leaf tea, La Tee Box includes a tea strainer with the first box as a Thank You. (La Tee Box 2016, 1.)

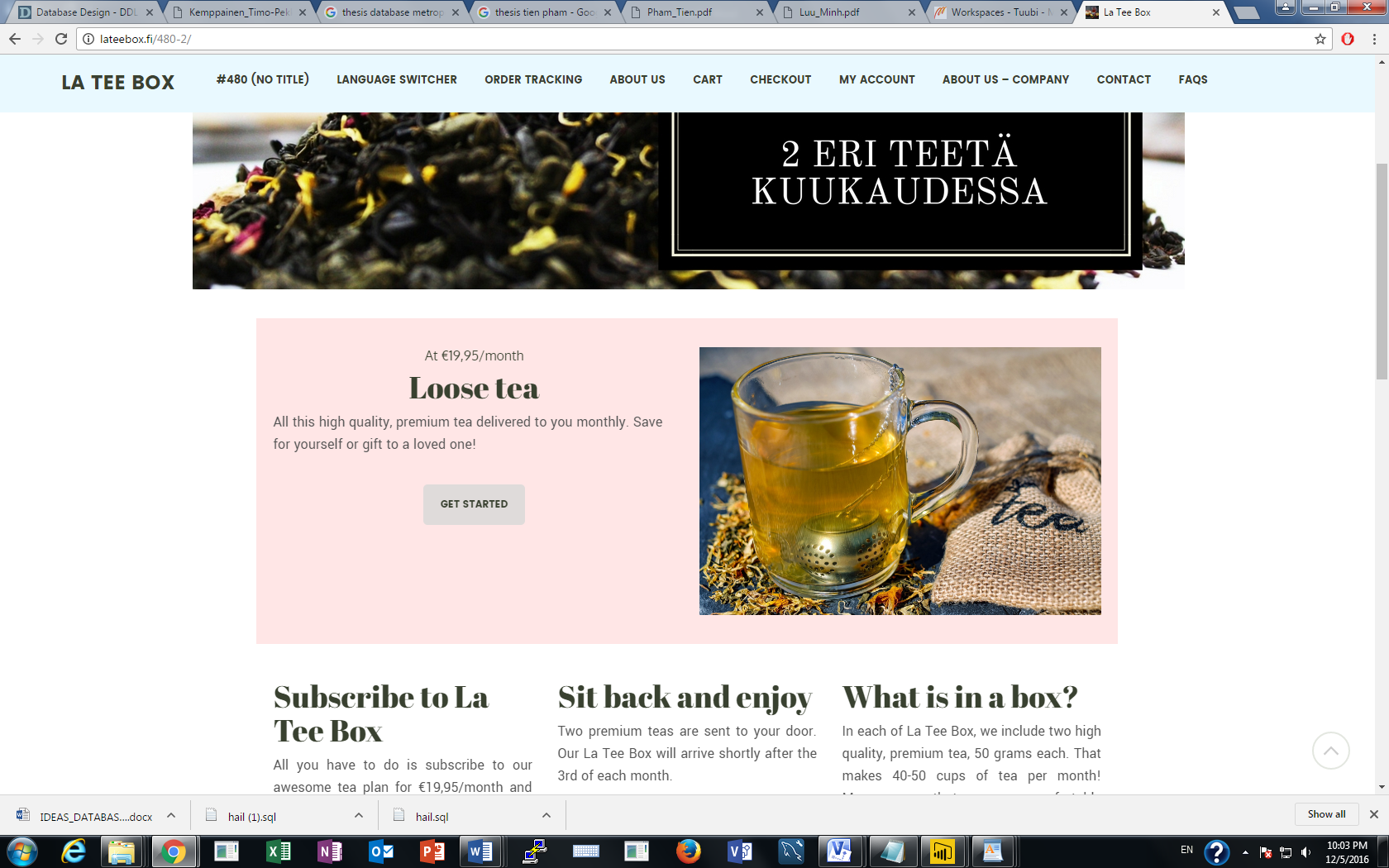


Figure 1: La Tee Box home page

Also, any interested person can subscribe to the monthly newsletter of the company by filling in their email information in the form in the following Figure 2:

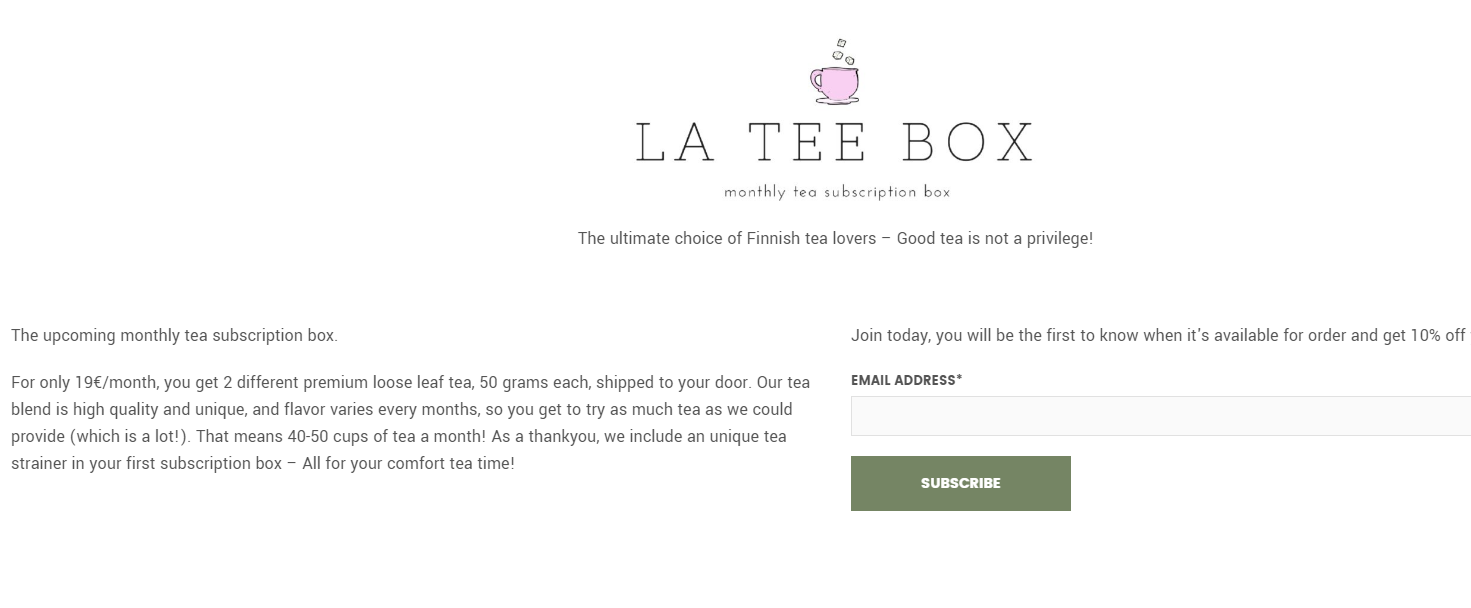


Figure 2: Monthly newsletter subscription form

## Steps of implementing the project

The project included several steps. First of all, an ER model was needed to graphically present the entities and the relations between them. In the second part, based on the ER model, the project team created a database by Database Definition Language (DDL) and Database Manipulation Language (DML), using PhpMyAdmin. Thirdly, some data was entered in the table so that queries can be run for database testing and trouble shooting. Finally, all of the data will be put into the database.

# User stories

As discussed previously, one of the team members is working for La Tee Box startup company. Therefore, the project team is able to communicate effectively with the users, which are La Tee Box owners. In the meeting where the team met the commissioning company, their expectations for the database became clear. They needed a convenient, easy tool to manage their data, as well as a way to calculate business information derived from the database. No security measures were really needed because of the local environment of the operation, since the company was small and everyone in the company had access to the database.

Therefore, the project team then created a relational database based on the requirements of the commissioning party. A relational [database](http://searchsqlserver.techtarget.com/definition/database) can be defined as collection of items organized as a set of formally-described [tables](http://searchsoa.techtarget.com/definition/table) from which data can be accessed or reassembled in many different ways without having to reorganize the database tables (Tech Target 2016, 1.).

Besides, the project team decided that the database management tool chosen for the database would be **PhpMyAdmin**. PhpMyAdmin allows users to use the database in a user friendly manner. Also, PhpMyAdmin was taught in the Metropolia’s database course that the authors were studying. A team member is working in the commissioning company so the database can be easily used and further implemented afterwards.

The team actively communicated with the company to give them updates about the project through the team member who was working there. La Tee Box allowed the project team to use **real data** of the company for product types and suppliers, but to protect the **confidentiality of customers**, customers’ names and orders would be just **random examples**.

# ER model

During the first step of database design, an ER-diagram is needed. ER-diagram was implemented to show the necessary data and the data type stored inside the database. Figure 3 describes the database’s ER-diagram. Each entity is a separate object described as a noun, such as Persons, Persons’ Type, Orders, Inventory, Category and Suppliers. Relationship represents how entities are related with each other and specified using Database Definition Language (DDL). When an entity joins a relationship, it is marked with a single line. For an identifying relationship, it is presented as a solid line, whereas for a non-identifying relationship or between strong entities it is marked with a dashed line. Each entity is represented as a table in ER-diagram. The attributes, which describe an entity, are placed inside the entity table. The NOT NULL attributes (the mandatory attributes of the table that cannot be left null) are formatted as **BOLD**.

As can be seen from Figure 3, the primary keys (PK) and foreign keys (FK) of the relations are specified. Relations between entities are handled using PK/FK mechanism. The foreign key attribute is posted by the ‘parent’ entity inside the ‘child’ table, using its own primary key. In an ER diagram, the primary keys are underlined.

From the beginning of the project, the project team wanted to create an ER model in Visio 2010 in order to use forward engineering function to generate SQL code. But after successfully doing so, it results in outdated codes that cannot be used, therefore the ER model generated by Visio is only used as a mean to explain the ER-diagram of the project.

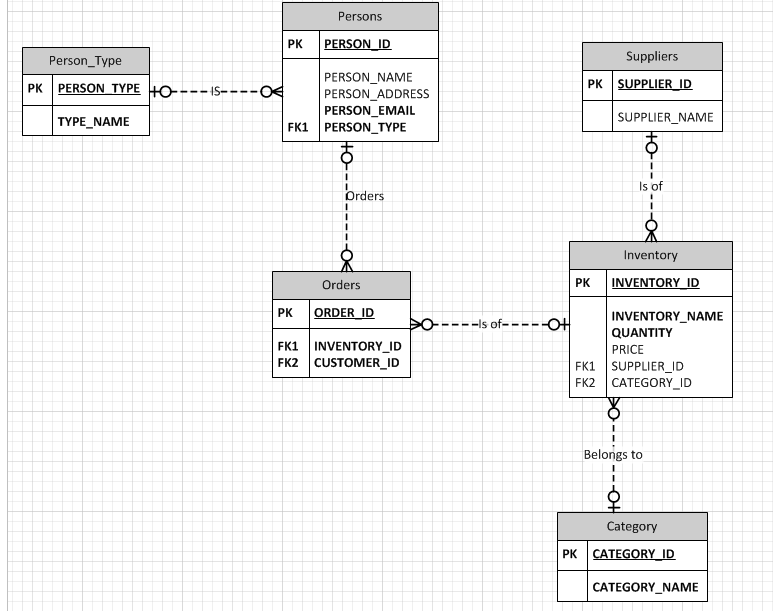


Figure 3: ER model created by Visio 2010

## Entities

Each table in the ER-diagram represents an entity with different attributes as well as foreign keys and/or primary key to specify its relationship with other tables. The purpose and idea of each table is presented in this sub part.

**Persons**

For La Tee Box database, it is needed to have a person to order, or to subscribe to the newsletter. There are three types of persons: buyer of tea or coffee, newsletter subscriber, and both buying products and subscribing. The types of person will be specified inside the Person\_Type table. Each person is given a unique id number which is PERSON\_ID, and the number is used to separate one person from others. After a person registers for newsletter or product subscription, that person will receive a thank you email for subscribing for the newsletter or an email with the order details. The primary key of this table is PERSON\_ID

In the Persons table, PERSON\_ID and PERSON\_TYPE cannot be null because those information identifies and classifies a customer and which type that customer belongs to. Therefore, they are NOT NULL. Likewise, the whole operation of the delivery service depends on customers’ email, the customer has to enter his or her email somewhere during the process, so customers’ email will also be NOT NULL. Other attributes such as PERSON\_ NAME and PERSON\_EMAIL address will only be required if the customer orders products, therefore those field are not mandatory.

**Person Type**

As mentioned above, each person will fall into one of three person types. The foreign key of Person\_Type table is Person\_Type. The Type\_name attribute will clarify the name of the person type. Both categories of this table is NOT NULL.

**Inventory**

Each type of inventory has a unique ID number which acts as the primary key of the Inventory table. Also, each inventory has a name, a quantity (Kilogram), the price for buying 1 kilogram of product from supplier (EUR). Inventory ID, NAME and QUANTITY are mandatory attributes which must be left NOT NULL. Two foreign keys, SUPPLIER\_ID and CATEGORY\_ID, are used to reference to Suppliers and Category tables where the product belongs to.

**Orders**

In reality, a person will place an order to receive a specific product, either tea or coffee. A person can order many types of products, and a product can be ordered by many persons. Therefore, the cardinality of the relationship between Persons and Inventory is N: M. In order to implement this relationship, the Orders table will act as a lookup table standing between the two tables. In this type of table implementing, the foreign key will in the ‘many’ side of the relationship. One person can place many orders, and one inventory type can appear in many orders. Therefore, the two foreign keys, INVENTORY\_ID and CUSTOMER\_ID, will be placed inside the Orders table.

Instead of a composite primary key with the two primary keys, the project team thought that it would be a much better idea to use a surrogate key so each customer can order many times in the Order table. That explains why ORDER\_ID is used as a primary key, instead of the combination of two foreign keys.

**Category and Suppliers**

These two tables are implemented in a similar fashion. The primary keys are the given unique ID of each entry, and a NAME column to specify the name of the entries. The team and the advising teacher discussed carefully whether Category table needed **a recursive relationship or not**. But the recursive relationship is not necessary in La Tee Box’s situation so the team did not implement it.

## Relations between entities

A brief explanation of the relationships between the entities:

- Persons have person type

- Persons order orders

- Inventory is of orders

- Order has Order details

- Inventory belongs to category

- Inventory belongs of suppliers

## Normalization

In order to minimize data redundancy and enhance data integrity, the database has been normalized. There are five normalization levels, which are First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), Fourth Normal Form (4NF), and Fifth Normal Form (5NF). However, there are a few redundancies that cannot be avoided to establish joints between entities. The database has been normalized and corrected properly with the guidance of the teacher using normalization technique.

## **Specifying data type of the attributes**

After successfully implementing the SQL queries and creating tables, the complete ER model created by PhpMyAdmin can be viewed as shown in Figure 4. For each table, ID is NOT NULL and AUTO\_INCREMENTED, so that the users do not need to enter ID for every entry and the ID will still be created. The majority of the columns are created with varchar(30) and int(11) which are enough, except the INVENTORY\_NAME with varchar(100) which contains many long product name. The PRICE and QUANTITY columns of Inventory table will be float to hold non-integers

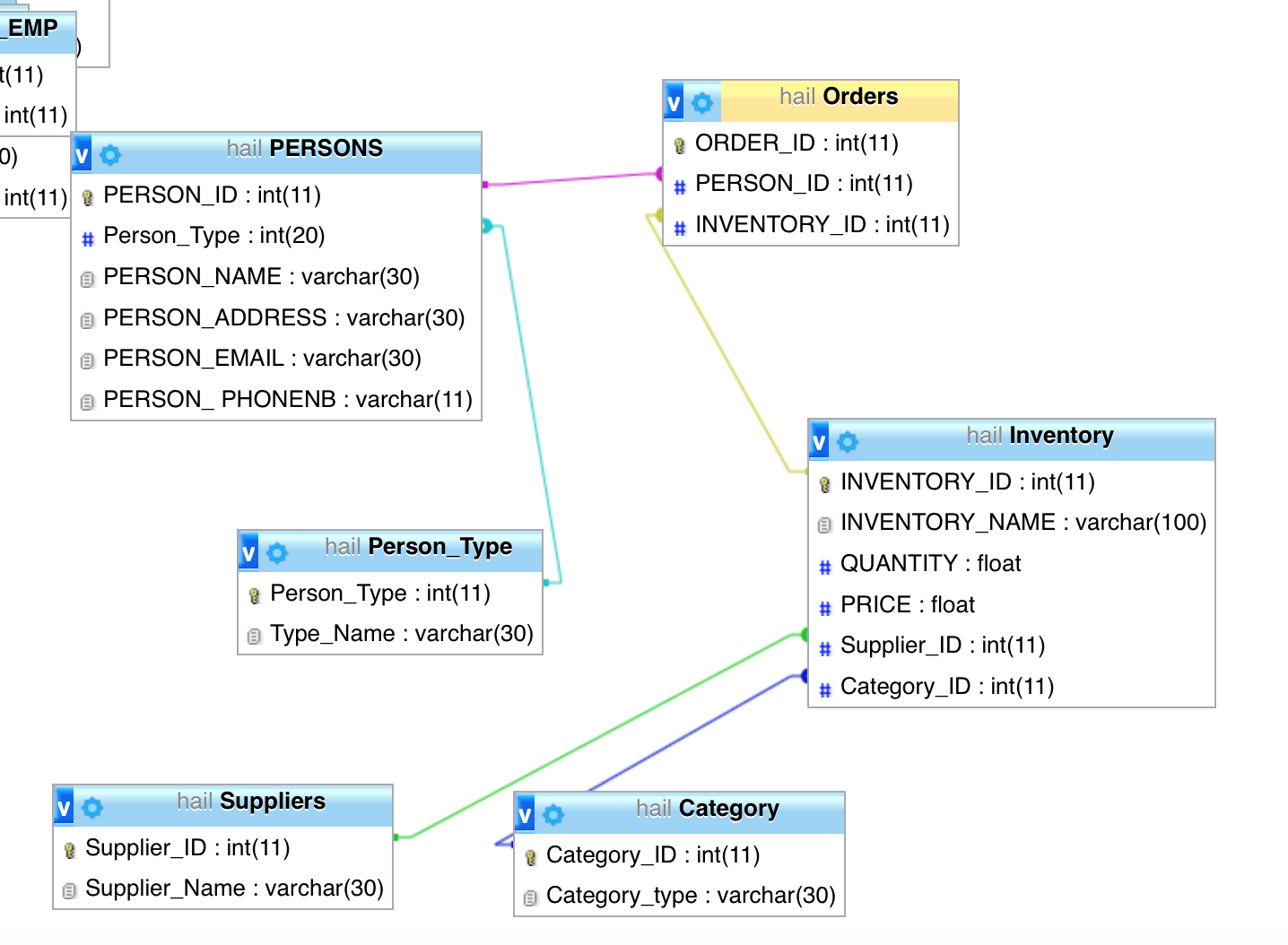


Figure 4. ER model generated by PhpMyAdmin

# DDL/SQL of the database

This project uses some basic types of Data Definition Language statements such as CREATE TABLE, ALTER TABLE, ADD KEY, MODIFY as below listings. They are highly useful functions that help authors to implement different kinds of tables as well as updating the database.

The storage engine used for the database is **INNODB.**

## CREATE

CREATE TABLE `Orders` (

`ORDER\_ID` int(11) NOT NULL,

`PERSON\_ID` int(11) NOT NULL,

`INVENTORY\_ID` int(11) NOT NULL

);

Listing 1. Create an orders table example

Orders table will mainly store the ID of order as well as person ID and inventory ID which derives from PERSONS table and INVENTORY TABLE.

Persons Table was designed and structured to thoroughly store useful information of person who place order the company main products tea and coffee.

CREATE TABLE `PERSONS` (

`PERSON\_ID` int(11) NOT NULL,

`Person\_Type` int(20) NOT NULL,

`PERSON\_NAME` varchar(30) DEFAULT NULL,

`PERSON\_ADDRESS` varchar(30) DEFAULT NULL,

`PERSON\_EMAIL` varchar(30) DEFAULT NULL,

`PERSON\_ PHONENB` int(11) DEFAULT NULL

);

Listing 2. Table design example

Because there are three types of customers, the person type needs to be classified. They maybe firm’s loyal customer, frequent buyers, or just some Newsletter Subscribers. The person type will be defined in the Person\_Type table creating by the query in Listing 3:

CREATE TABLE `Person\_Type` (

`Person\_Type` int(11) NOT NULL,

`Type\_Name` varchar(30) NOT NULL

);

Listing 3. Person type Table

Toward supplier database, a Suppliers table is created to manage list of suppliers in terms of coffee and tea products.

CREATE TABLE `Suppliers` (

`Supplier\_ID` int(11) NOT NULL,

`Supplier\_Name` varchar(30) NOT NULL

);

Listing 4. Suppliers Table

Similarly, Category table and its data added in following listing 5:

CREATE TABLE `Category` (

`Category\_ID` int(11) NOT NULL,

`Category\_type` varchar(30) NOT NULL

);

Listing 5. Catergory implementation

## ALTER

The ID attributes of all tables in the database are AUTO\_INCREMENTED so that the users do not need to enter an ID for each of them, and each attribute will be unique. The SQL code for this is:

ALTER TABLE `Category`

MODIFY `Category\_ID` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=3;

Listing 6: Auto incremented ID attribute

KEY (includes PRIMARY KEY) also add to each table for their unique identifications:

ALTER TABLE `Category`

ADD PRIMARY KEY (`Category\_ID`);

Listing 7. Create primary key as CATEGORY\_ID

Inventory adding example:

ALTER TABLE `Inventory`

ADD PRIMARY KEY (`INVENTORY\_ID`),

ADD KEY `Supplier\_ID` (`Supplier\_ID`),

ADD KEY `Category\_ID` (`Category\_ID`);

Listing 8. Alter table Inventory to add primary key for INVENTORY\_ID and other keys

In Listing 9, constraints of table Orders are implemented. One important point to note is that in relational database, the tables are linked with constraints. It is necessary to use **CASCADE** so that the other tables are up to date when a change happens in one table. CASCADE specifies that the column will be updated when the referenced column is updated, and rows will be deleted when the referenced rows are deleted (SQL and Me 2016, 1.).

ALTER TABLE `Orders`

ADD CONSTRAINT `Orders\_ibfk\_2` FOREIGN KEY (`INVENTORY\_ID`) REFERENCES `Inventory` (`INVENTORY\_ID`) ON UPDATE CASCADE,

ADD CONSTRAINT `Orders\_ibfk\_3` FOREIGN KEY (`PERSON\_ID`) REFERENCES `PERSONS` (`PERSON\_ID`) ON UPDATE CASCADE;

Listing 9: Adding constraints using UPDATE CASCADE

## DROP

At several points of the project, the authors decided that certain tables, or constraints were completely unnecessary. Thanks to the DROP statement of SQL, authors avoided messing things up, which is a time consuming process in database management.

Following Listing present how DROP function actually help authors get rid of data base objects.

DROP TABLE category;

DROP INDEX ID on TABLE category;

ALTER TABLE `Inventory`

ADD CONSTRAINT `Inventory\_ibfk\_1`

Listing 10: Dropping table and constraint

# DML/SQL of CRUD + SELECT examples

One of the main crucially important functions of SQL is the ability of this language in manipulate database system. In the following examples , the project continues to give readers the general view of SQL manipulation functions.

## INSERT

Table without detail database is useless in database management, the important of database inside every table is that it makes clear what actually the data should be. Besides, data needs to be added for error trouble-shooting. Indeed, by inserting detail information of each person into Persons table, the data is now human-friendly and approachable. The following listing is an example how SQL code can be used to insert the data into Persons Table.

INSERT INTO `PERSONS` (`PERSON\_ID`, `Person\_Type`, `PERSON\_NAME`, `PERSON\_ADDRESS`, `PERSON\_EMAIL`, `PERSON\_ PHONENB`) VALUES

(1, 1, 'Timo Leinonen', 'Koskelantie 56 B 1', 'timoleino@gmail.com', NULL),

(2, 3, 'Laura Nurminen', 'Antti Korpin Tie 4 c 3', 'habet@gmail.com', NULL),

(3, 2, NULL, NULL, 'helihu@gmail.com', NULL),

(4, 2, NULL, NULL, 'nifagerit@gmail.com', NULL);

(5, 1, 'Heidi Kotiranta', 'kuohujuja 6E83', 'alexstubb@gmail.com', '0403234511'),

(6, 2, 'Saari Lahtinen', 'Sahratie 6, Vantaa', 'saari.lahtinen@gmail.com', '0431245655'),

(7, 2, 'Hella Kasvinen', 'Kivirinne 8B', 'hella.kasvinen@yahoo.com', '0450675231'),

Listing 11. Adding example data in to PERSONS table

As discussed in ER model chapter, not all person requires a name or an address. The reason for that is because some are only newsletter subscribers, therefore the company does not have access to their names and address. In Listing 11, persons with ID 3 and 4 are examples of persons without name and address.

In order to manage all potential customers, authors decide to divide person type into to 3 sub-type, respectively, loyal/monthly subscribers, potential buyers, and newsletters subscribers.

INSERT INTO `Person\_Type` (`Person\_Type`, `Type\_Name`) VALUES

(3, Loyal/monthly subscribers),

(1, 'potential buyers'),

(2, 'Newsletter Subscribers');

Listing 12. Adding customer types into person type table

Additionally, adding data into Suppliers table, currently company has 5 main suppliers as following;

INSERT INTO `Suppliers` (`Supplier\_ID`, `Supplier\_Name`) VALUES

(1, 'Jenier World of Teas'),

(2, 'Tetley'),

(3, 'Yorkshire'),

(4, 'Philips Senseo '),

(5, 'Taylors of Harrogate');

Listing 13. Values inserted into Supplier table

There are only two kinds of products that company is selling:

INSERT INTO `Category` (`Category\_ID`, `Category\_type`) VALUES

(1, 'Tea'),

(2, 'Coffee');

Listing 14. Insert values into category table.

Below is an example of updating the price of one item in inventory Table when suppliers change price.

BEFORE:

PRICE = 12.5



Figure 5. price before change

AFTER:

UPDATE Inventory SET PRICE = 11.5 WHERE INVENTORY\_NAME = ‘China Congou Mao Jia Green Tea’;



Figure 6. Price after change

/\*SORT CUSTOMER NAME IN ORDER BY NAME ASCENDING\*/

SELECT \* FROM PERSONS ORDER BY PERSON\_NAME ;

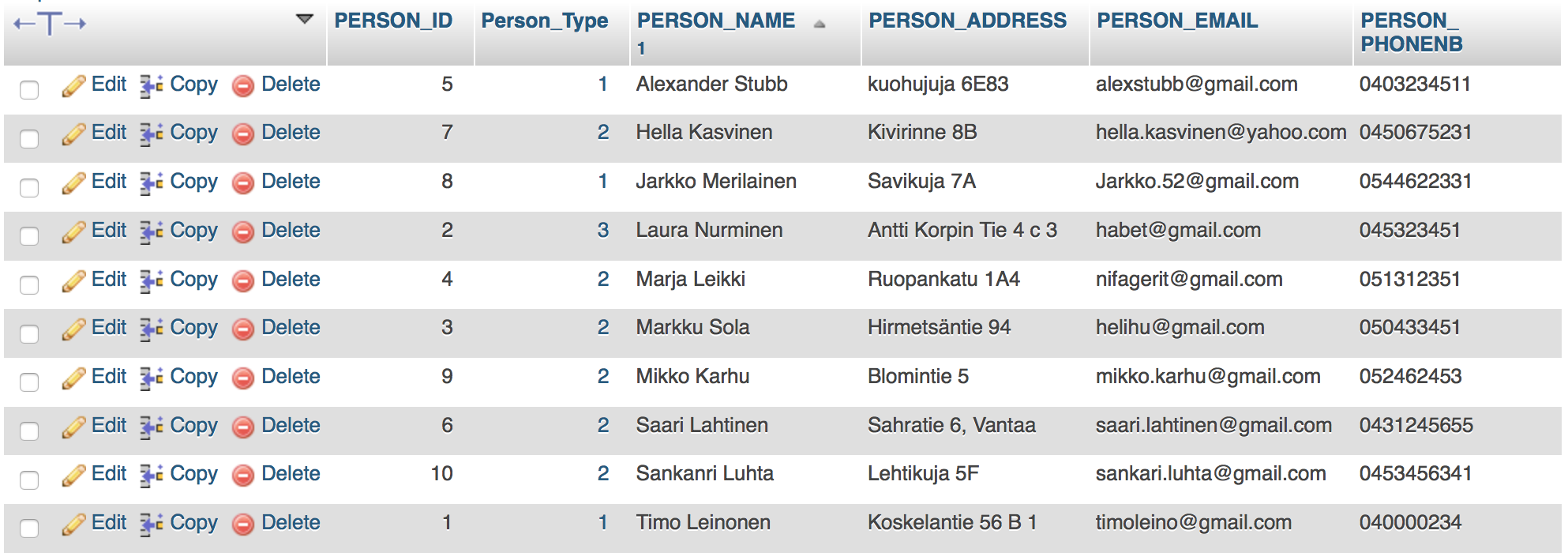


Figure 7. Sorting customer name

## UPDATE

The update statement is used to change values that are already in a table. In this project, there is often the practical situation in which customers often change their order detail after a period time of trying and tasting some brand of tea or coffee. Therefore, UPDATE statement appears to be a very useful statement in order to handling the ever-changing database.

For instance, authors need to change the product ordered by the 3rd person from Persons table from currently INVENTORY\_ID 3 AND 5 into INVENTORY\_ID = 8;

Code:

UPDATE Orders SET INVENTORY\_ID = 8 WHERE PERSON\_ID = 3

Before changed making:

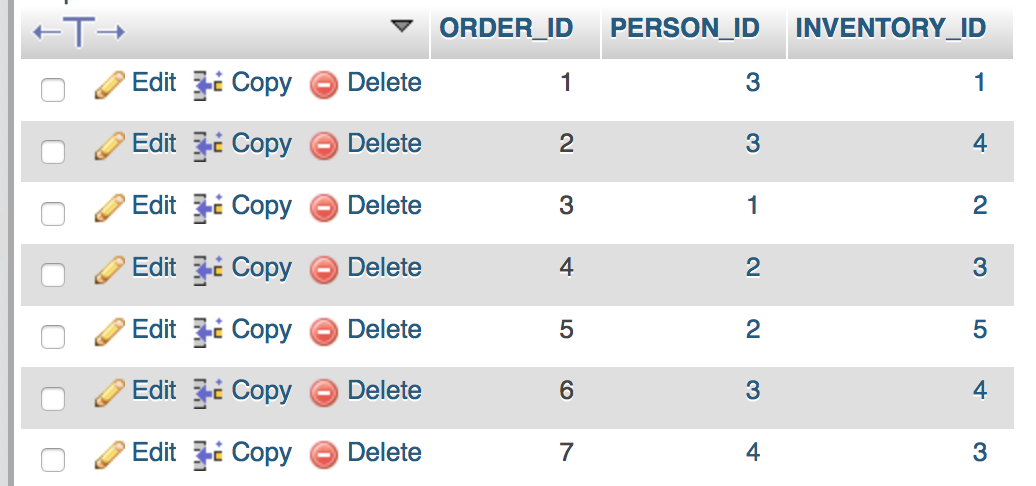


Figure 8. Order Table before inventory changing

After changed making:

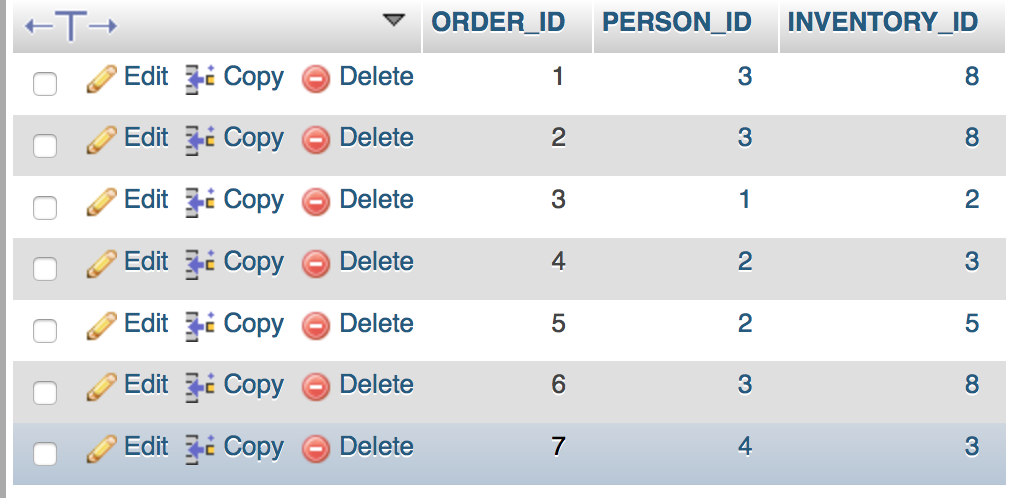


Figure 9. Order Table after inventory changing

## DELETE

The SQL code for deleting an entry is:

/\*DELETE ONE PERSON WHO IS NO LONGER SUBCRIBE\*/

DELETE FROM INVENTORY WHERE INVENTORY\_ID = "7";

As mentioned in sub part 4.2, keeping the constraints **CASCADE** is important to keep all the tables up to date. Besides, it is also required if user wants to delete a certain entry in a linked table. An error will occur if CASCADE was not enabled. In the following figure, the authors attempted to delete row number 7 without enabling CASCADE.

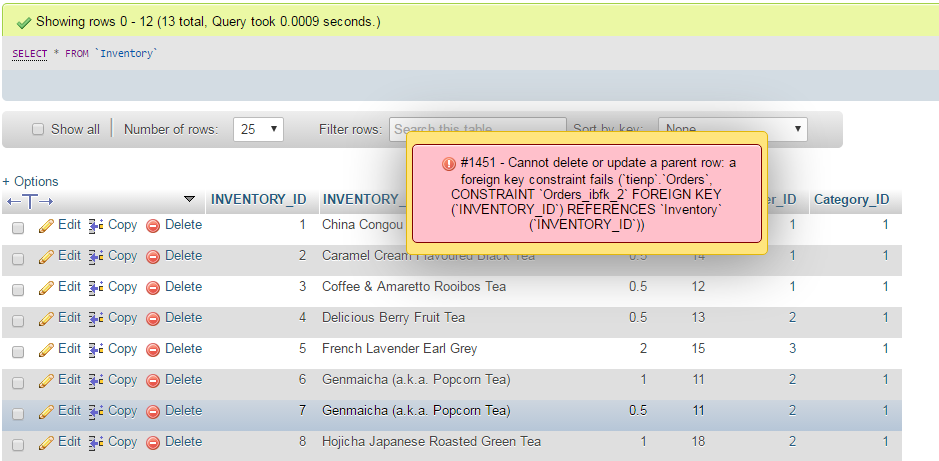
****

Figure 10: Failed attempt to delete entry in linked table

The result is that the entry cannot be deleted. Acknowledging the problem, the project team changed the constraint into UPDATE CASCADE using the code in Listing 14.

ALTER TABLE `Orders`

ADD CONSTRAINT `Orders\_ibfk\_2` FOREIGN KEY (`INVENTORY\_ID`) REFERENCES `Inventory` (`INVENTORY\_ID`) ON UPDATE CASCADE,

ALTER TABLE `Inventory`

ADD CONSTRAINT `Inventory\_ibfk\_1` FOREIGN KEY (`Supplier\_ID`) REFERENCES `Suppliers` (`Supplier\_ID`) ON UPDATE CASCADE,

ADD CONSTRAINT `Inventory\_ibfk\_2` FOREIGN KEY (`Category\_ID`) REFERENCES `Category` (`Category\_ID`) ON UPDATE CASCADE;

Listing 15: UPDATE CASCADE CONSTRAINT to enable deleting entries

Now row number 7 was deleted successfully:

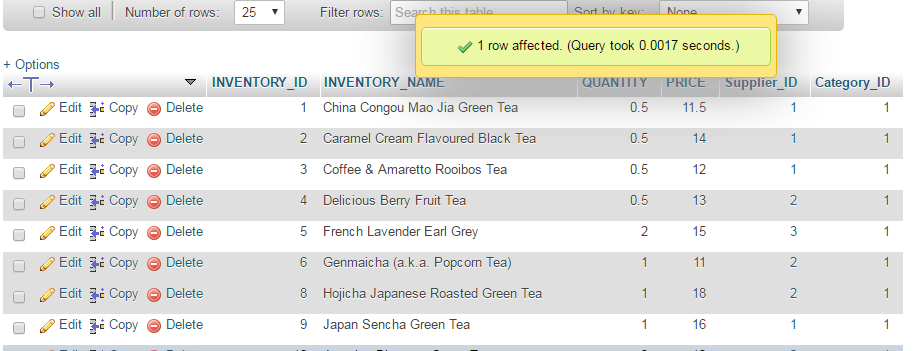
****

Figure 11: Successful attempt to delete entry after enable UPDATE CASCADE

## **SELECT examples**

In this example, the authors wanted to count the number of people ordering Rooibos Tea.

/\*SELECT how many people order Coffee & Amaretto Rooibos Tea\*/

SELECT COUNT(INVENTORY\_ID) AS RESULT FROM ORDER WHERE INVENTORY\_ID = 3;

ORDER TABLE:

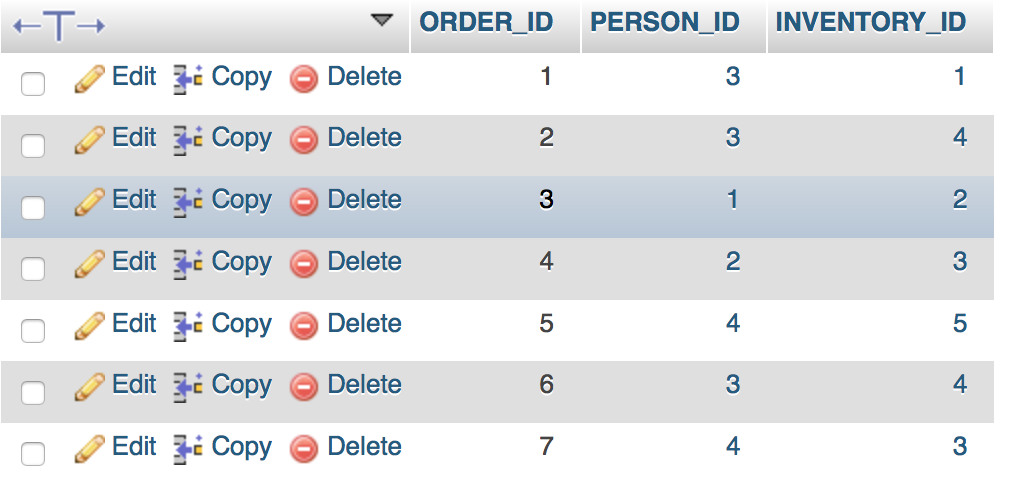


Figure 12. Inventory table

RESULT:

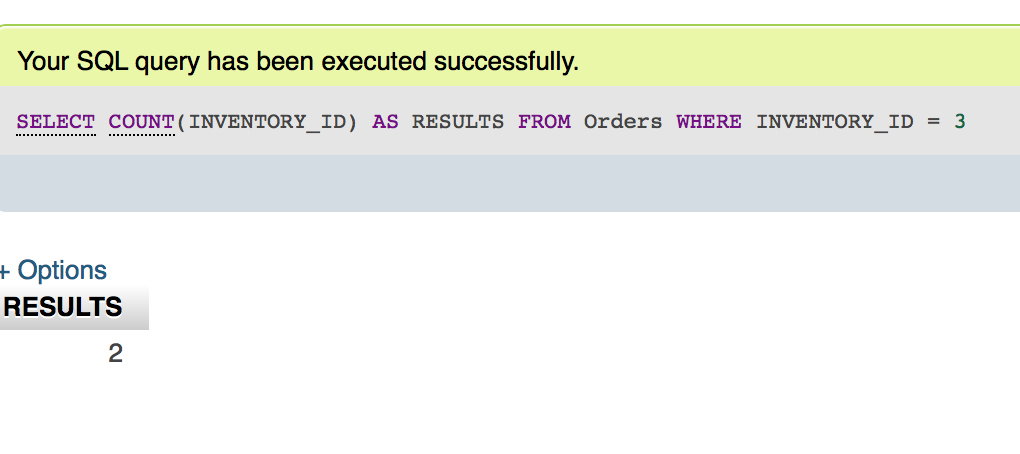


Figure 13. Counting inventory ”Coffee & Amaretto Rooibos Tea”

/\*SELECT INVENTORY WHERE PRICE IS LESS THAN 12 EURO \*/

SELECT \* FROM INVENTORY WHERE PRICE < 12;

RESULT:



Figure 14. Price sorting less than 12 euros

/\*COUNT SUM OF ORDER \*/

SELECT COUNT(ORDER\_ID) AS TotalItemOrder FROM Orders;

RESULT:

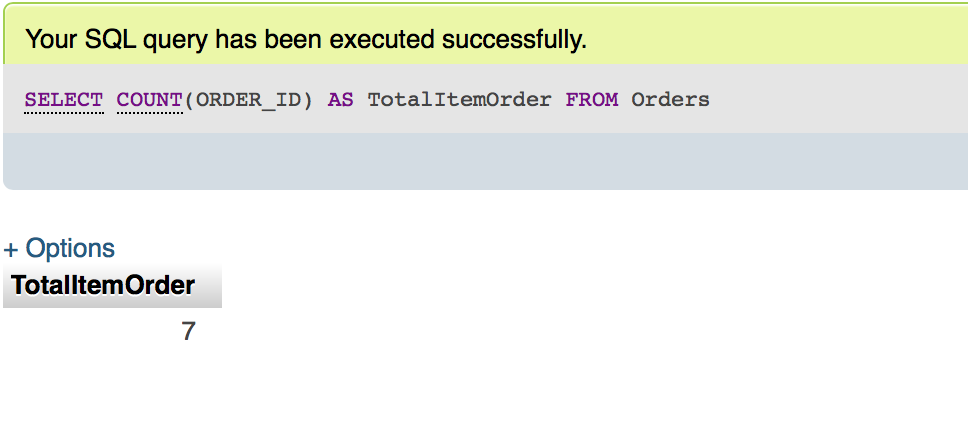


Figure 15. Sum of orders

/\* Select and calculate total value of each order \*/

SELECT Orders.ORDER\_ID, SUM(Inventory.PRICE) AS totalRevenue

FROM Orders, Inventory

WHERE Orders.INVENTORY\_ID = Inventory.INVENTORY\_ID

GROUP BY Inventory.INVENTORY\_ID

The result after that is:

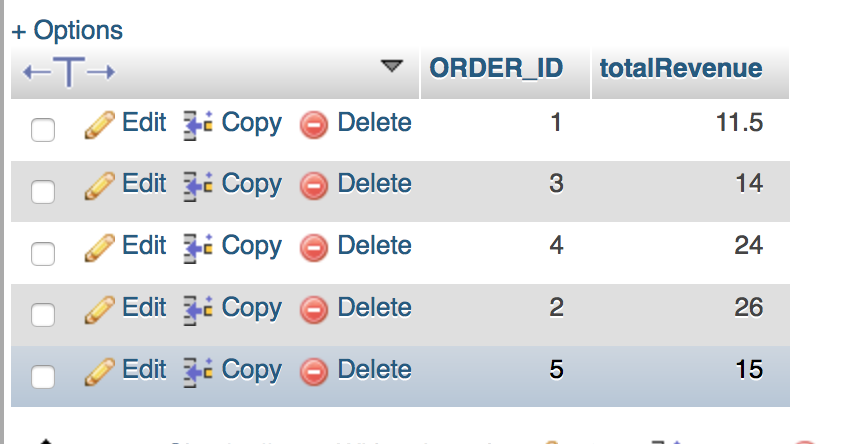


Figure 16. Price per order / revenue when selling

# Conclusion

The project is proved to be useful to the end users, which are the owners of La Tee Box. The end users now have a mean to manage data in stead of traditionally handling data in MS Excel, data is now more readable. After the project, the database became highly managable and easy to manipulate.

On the other hand, the project also fulfills data management course ’s final data project requirement. It is said that learning from practical examples plays an important role in learning process. The project team gained more knowledge and experiences in data management of small scale business at basic level.

During the project, the team had the opportunity to practice and learn by mistakes. Certain problems occurred, such as not being able to delete an entry without CASCADING, but the team together overcame those issues. By praticing with the database daily, the team members believe that their data management skills will be gradually improved in the future.

Finally, data management and especially SQL, proved to be important. Knowing the basis of SQL, a person can use any database in the world with little or no learning curve.

# Electronic references

La Tee Box. 2016. La Tee Box home page.

<http://lateebox.fi/480-2/> Accessed: 5 Dec 2016

SQL and Me. 2016. SQL Server – How to cascade UPDATEs and DELETEs to related tables

<https://sqlandme.com/2011/08/08/sql-server-how-to-cascade-updates-and-deletes-to-related-tables/> Accessed: 6 Dec 2016

Tech Target. 2016. Relational Database

<http://searchsqlserver.techtarget.com/definition/relational-database> Accessed: 1 Dec 2016