INFSCI 2710 Database management

Final Group Project Report

December 1, 2014

Qilu Cao (QIC27), wanzhi hu (wah16), haoda zou (haz51)

Contents

[Team members 1](#_Toc405219369)

[Project Description 1](#_Toc405219370)

[Overview 1](#_Toc405219371)

[Assumptions 1](#_Toc405219372)

[Entity-Relation Diagram 1](#_Toc405219373)

[DDL Statement 4](#_Toc405219374)

[Description of front-end to back-end connection 5](#_Toc405219375)

[Implementation 6](#_Toc405219376)

[Testing efforts 10](#_Toc405219377)

[Limitations and improvements 11](#_Toc405219378)

# Team members

Qilu Cao (QIC27@pitt.edu)

Wanzhi Hu (WAH16@pitt.edu)

Haoda Zou (HAZ51@pitt.edu)

# Project Description

A Digital Sales Corporation has about 100 stores in different regions with average of 1000 customers per store. To perform more efficiently, Digital Sales decided to develop a database systems to run day-to-day transactions of a company. Such system is also referred to as on-line transaction processing (OLTP) systems, where the database is updated continually throughout the day. The transactions should be captured at the point of sales, one row of the transaction table is used for each purchased item.

# Overview

This system allows users to view product lists and purchase history, allows employees to add or sell existing products, and allows administrator to add, edit or delete most of the information. It is accessible to customers, employees and database administrator. It also allows those users to search and sort the results in a flexible way. In addition, the system is provide a logging feature for users to see their operation history.

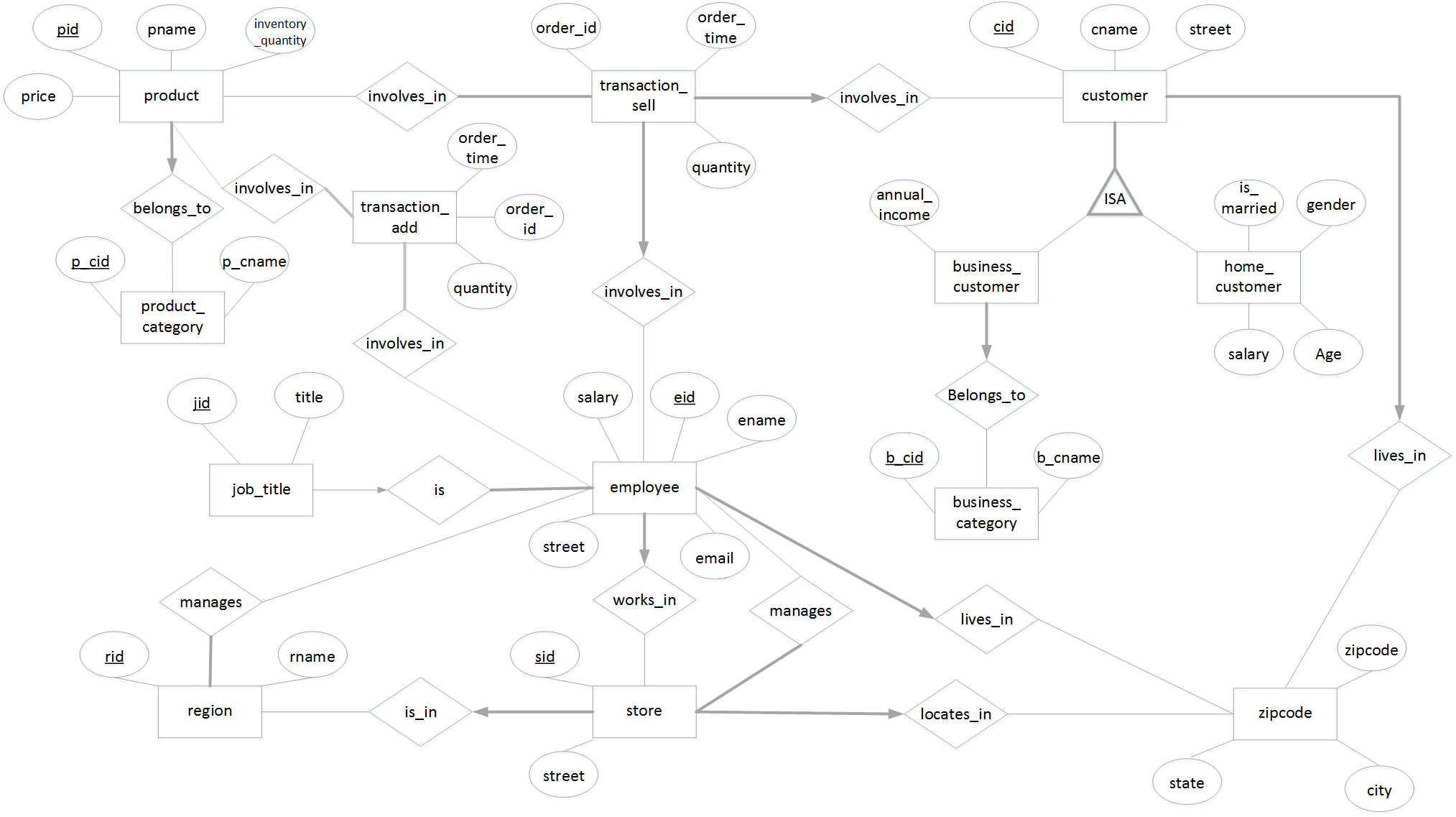
# Assumptions

We assume that the system has the following functions:

1. Employee can use system to add transaction when she/he add products or sells products to one customer.
2. Customers can see the details of products and their purchase history in database.
3. Administrator can modify most of the data
4. Users may need to search or sort different attributes of data in every table.
5. All users can see the sales report implemented by aggregation.
6. Administrator may want to see the log to know what happens to the database system.

# Entity-Relation Diagram

The following graph is the E-R diagram that we designed.



There are totally 15 entities in this database system:

1. **zipcode**: This is an imported table with all the zip code related data of the U.S. The attributes include **zipcode,** **city** and **state**. The primary key is **zipcode**.
2. **customer**: The attributes are customer ID (**cid**), customer name (**cname**), zipcode and street. The primary key is **cid**, the foreign key is **zipcode**, which references **zipcode**.
3. **home\_customer**: Is a subset of customer, its own attributes are **is\_married**, **gender**, **age** and **salary**. The primary key is **cid**, the foreign key is also **cid** which references customer.
4. **business\_customer**: Is another subset of customer, and its own attributes are annual income and business category. The **business\_customer** depends on **business\_category**. The primary key is **cid**, the foreign keys are **cid** which references customer, and **b\_cid** which references **business\_category**.
5. **business\_category**: the arrtibutes are business category ID(**b\_cid)**, business category name(**b\_cname**). the primary key is **b\_cid**.
6. **transact\_sell**: lists all the selling activities. The attributes are **order\_number**, **order\_date**, **quantity**, **price**, customer ID(**cid)**, product id(**pid**) and employee id(**eid**). The primary key is **order\_number**, the foreign key is **cid**, **pid**, and **eid** which references customer, product and employee respectively.
7. **transact\_add**: lists all the stock adding activities. The attributes are **order\_number**, **order\_date**, **quantity**, **price**, product id(**pid**) and employee id(**eid**). The primary key is **order\_number**, the foreign key is **pid**, and **eid** which references product and employee.
8. **product**: the attributes are product id(**pid**), product name(**pname**), **price, quantity** and product category id(**p\_cid**). The primary key is **pid**, the foreign key is **p\_cid** which references **product\_category**.
9. **product\_category**: the attributes are product category id (**p\_cid**) and product category name(**p\_cname**). The primary key is **p\_cid**.
10. **employee**: describes the information of all the employees in the digital sales corporation. The attributes are employee id (**eid**), employee name(**ename**), salary, job title id(**jid**), **email**, **zipcode** and store id(**sid**). The primary key is **eid**, the foreign key is **sid** and **zipcode**, which references store and **zipcode**.
11. **job\_title**: the attributes are job title id (**jid**) and job **title**. The primary key is **jid**.
12. **store:** the attributes are store id(**sid**), **zipcode, street,** and region id(**rid**). The primary key is **sid**, the foreign key is **rid** which references **region**.
13. **store\_manager**: the attributes are employee id (**eid**) and store id (**sid**). The primary key is sid, the foreign keys are **eid** references employee and **sid** references store. Here we assume that one store manager can only manages at most one store.
14. **region:** the attributes are region id(**rid**) and region name(**rname**). The primary key is **rid**.
15. **region\_manager**: the attributes are employee id(**eid**) and region id(**rid**). The primary key is **rid**, the foreign key is **eid** references employee. Here we assume that one region manager can only manages at most one region.

# DDL Statement

Here are the statements for creating tables:

CREATE TABLE zipcode(zipcode CHAR(5) NOT NULL, city CHAR(20) NOT NULL, state CHAR(2) NOT NULL, PRIMARY KEY(zipcode));

CREATE TABLE customer(cid INTEGER(8) Zerofill AUTO\_INCREMENT, cname CHAR(20) NOT NULL, zipcode CHAR(5) NOT NULL, street CHAR(40) NOT NULL, PRIMARY KEY(cid), FOREIGN KEY(zipcode) REFERENCES zipcode(zipcode));

CREATE TABLE business\_category(b\_cid INTEGER(8) Zerofill AUTO\_INCREMENT, b\_cname CHAR(40) NOT NULL, PRIMARY KEY(b\_cid));

CREATE TABLE business\_customer(cid INTEGER(8) Zerofill NOT NULL, annual\_income REAL NOT NULL, b\_cid INTEGER(8) Zerofill NOT NULL, PRIMARY KEY(cid), FOREIGN KEY(cid) REFERENCES customer(cid) ON DELETE CASCADE, FOREIGN KEY(b\_cid) REFERENCES business\_category(b\_cid));

CREATE TABLE home\_customer(cid INTEGER(8) Zerofill NOT NULL, is\_married TINYINT(1) NOT NULL, gender TINYINT(1) NOT NULL, age INTEGER(3) NOT NULL, salary REAL NOT NULL, PRIMARY KEY(cid), FOREIGN KEY(cid) REFERENCES customer(cid) ON DELETE CASCADE);

CREATE TABLE product\_category(p\_cid INTEGER(8) Zerofill AUTO\_INCREMENT, p\_cname CHAR(40) NOT NULL, PRIMARY KEY(p\_cid));

CREATE TABLE product(pid INTEGER(8) Zerofill AUTO\_INCREMENT, pname CHAR(20) NOT NULL, price REAL NOT NULL, p\_cid INTEGER(8) Zerofill, quantity INTEGER(8) NOT NULL, PRIMARY KEY(pid), FOREIGN KEY(p\_cid) REFERENCES product\_category(p\_cid));

CREATE TABLE region(rid INTEGER(8) Zerofill AUTO\_INCREMENT, rname CHAR(40) NOT NULL, PRIMARY KEY(rid));

CREATE TABLE store(sid INTEGER(8) Zerofill AUTO\_INCREMENT, zipcode CHAR(5) NOT NULL, street CHAR(40) NOT NULL, rid INTEGER(8) Zerofill NOT NULL, PRIMARY KEY(sid), FOREIGN KEY(rid) REFERENCES region(rid), FOREIGN KEY(zipcode) REFERENCES zipcode(zipcode));

CREATE TABLE job\_title(jid INTEGER(8) Zerofill AUTO\_INCREMENT, title CHAR(40) NOT NULL, PRIMARY KEY(jid));

CREATE TABLE employee(eid INTEGER(8) Zerofill AUTO\_INCREMENT, ename CHAR(20) NOT NULL, salary REAL NOT NULL, jid INTEGER(8) Zerofill NOT NULL, email CHAR(30) NOT NULL, zipcode CHAR(5) NOT NULL, street CHAR(40) NOT NULL, sid INTEGER(8) Zerofill NOT NULL, PRIMARY KEY(eid), FOREIGN KEY(jid) REFERENCES job\_title(jid), FOREIGN KEY(sid) REFERENCES store(sid), FOREIGN KEY(zipcode)REFERENCES zipcode(zipcode));

CREATE TABLE transact\_add(order\_id INTEGER(8) Zerofill AUTO\_INCREMENT, order\_time TIMESTAMP, quantity INTEGER(8) NOT NULL, pid INTEGER(8) Zerofill NOT NULL, eid INTEGER(8) Zerofill NOT NULL, PRIMARY KEY(order\_id), FOREIGN KEY(eid) REFERENCES employee(eid), FOREIGN KEY(pid) REFERENCES employee(pid));

CREATE TABLE transact\_sell(order\_id INTEGER(8) Zerofill AUTO\_INCREMENT, order\_time TIMESTAMP, quantity INTEGER(8) NOT NULL, cid INTEGER(8) Zerofill NOT NULL, pid INTEGER(8) Zerofill NOT NULL, eid INTEGER(8) Zerofill NOT NULL, PRIMARY KEY(order\_id), FOREIGN KEY(cid)REFERENCES customer(cid), FOREIGN KEY(pid)REFERENCES product(pid), FOREIGN KEY(eid) REFERENCES employee(eid));

CREATE TABLE store\_manager(eid INTEGER(8) Zerofill NOT NULL, sid INTEGER(8) Zerofill NOT NULL, PRIMARY KEY(sid), FOREIGN KEY(eid)REFERENCES employee(eid), FOREIGN KEY(sid)REFERENCES store(sid));

CREATE TABLE region\_manager(eid INTEGER(8) Zerofill, rid INTEGER(8) Zerofill NOT NULL, PRIMARY KEY(rid), FOREIGN KEY(eid)REFERENCES employee(eid), FOREIGN KEY(rid)REFERENCES region(rid));

We apply the ZEROFILL and AUTO\_INCREMENT function to each ID attribute, which make it easier to manage the ID. We also use TIMESTAMP to manage order\_time automatically.

Log system was created and operated in a different database to reduce the risk of crash:

CREATE TABLE log(logtime TIMESTAMP, is\_error TINYINT(1), detail TEXT);

# Description of front-end and back-end design

The front-end of the system is a user-friendly website. There is a navigation bar at the top of page for user to choose which data they want to manage. The body of the page lists data with table. The user can edit, delete, search, sort or add tuples in a same table, which can increase productivity significantly.

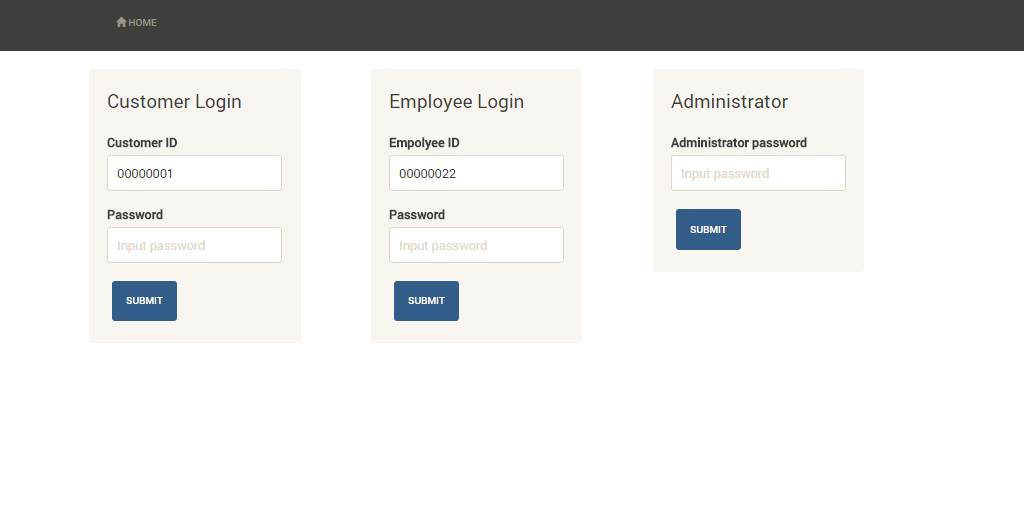
On the right of navigation bar, users can click “view log” link to see SQL statement they have executed to database.

The front-end of system is implemented by using AJAX technology. GET is used for querying and listing results, and the cache will increase speed when refreshing. POST is used for adding, updating and deleting for safety concern. PHP scripts focus on generating SQL statements, connecting MySQL, executing queries and returning results in JSON format. Any exception will not be exposed directly on the front-end. Instead the front-end will disallow any further actions and suggest users to redirect to the log viewer.

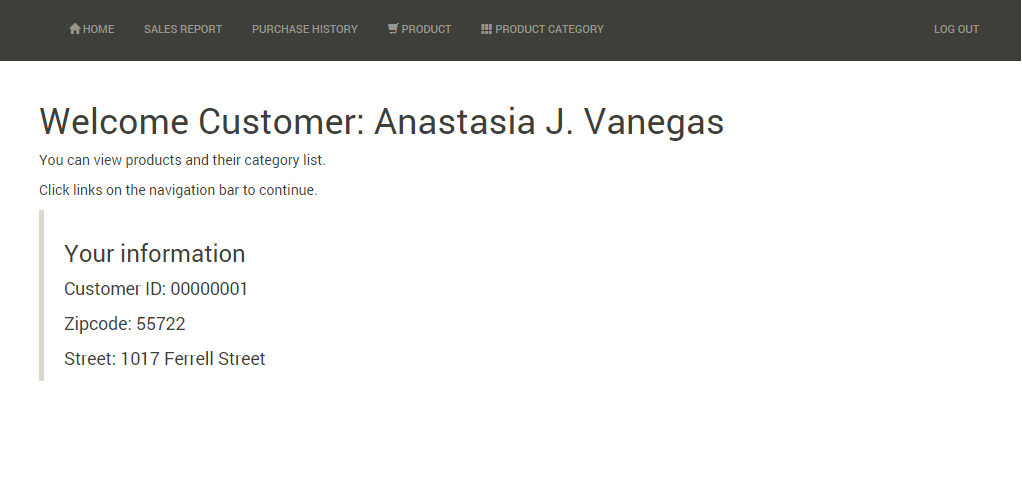
# Implementation

Here shows how products management is implemented:

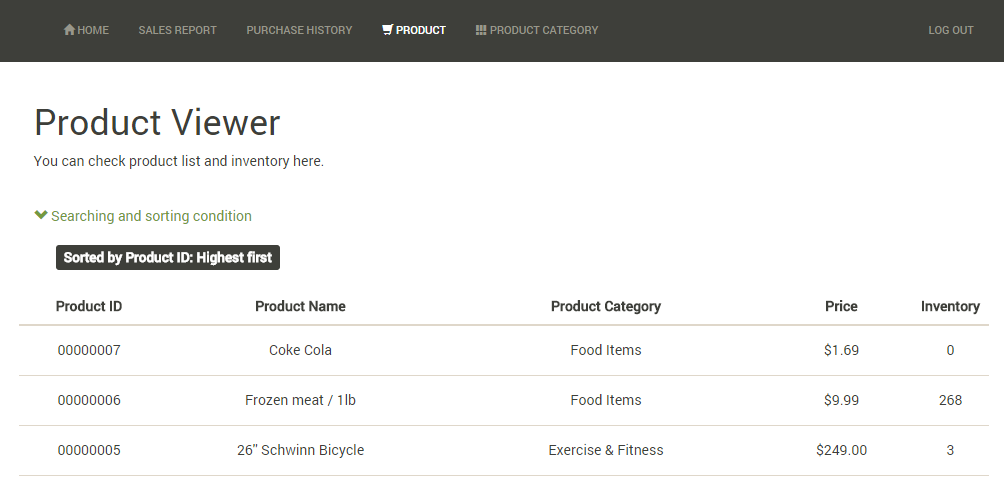
1. Login Interface



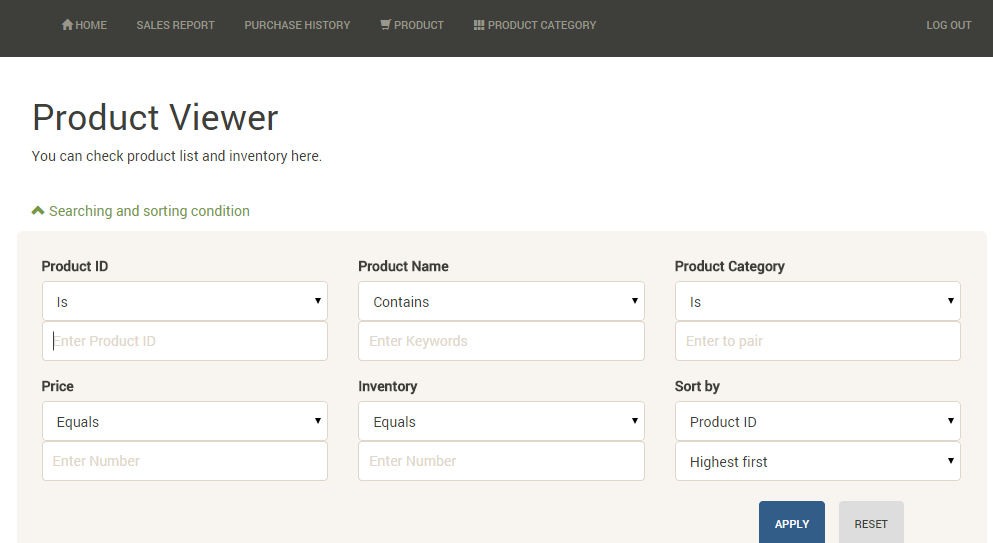
1. After login, there will have different links available for different users.



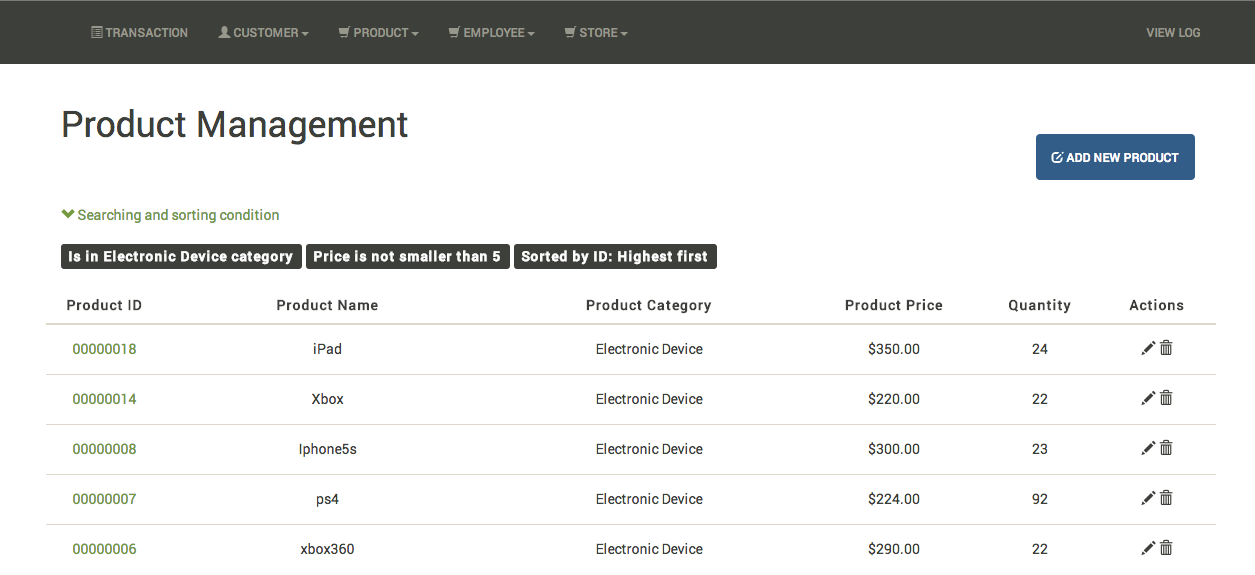
1. This is users’ view of product list. Other read-only mode are similar

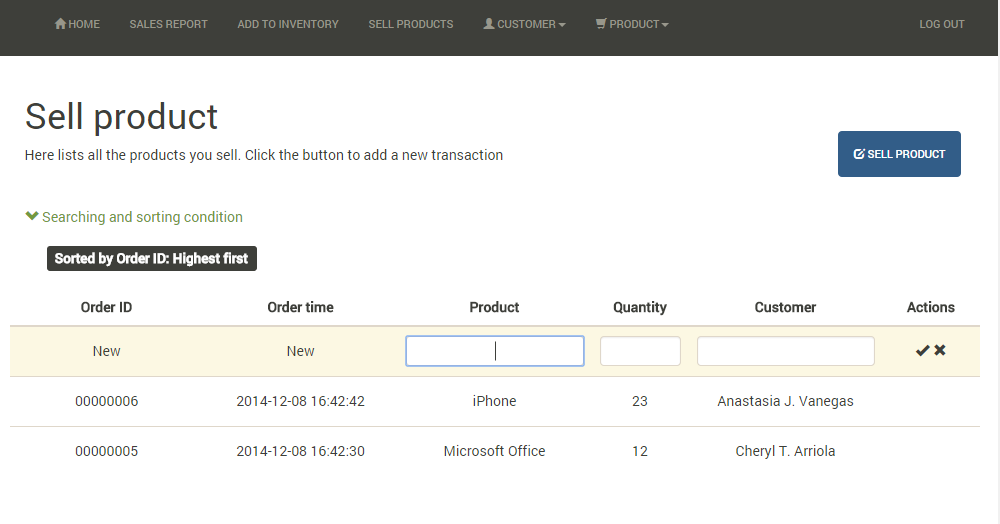
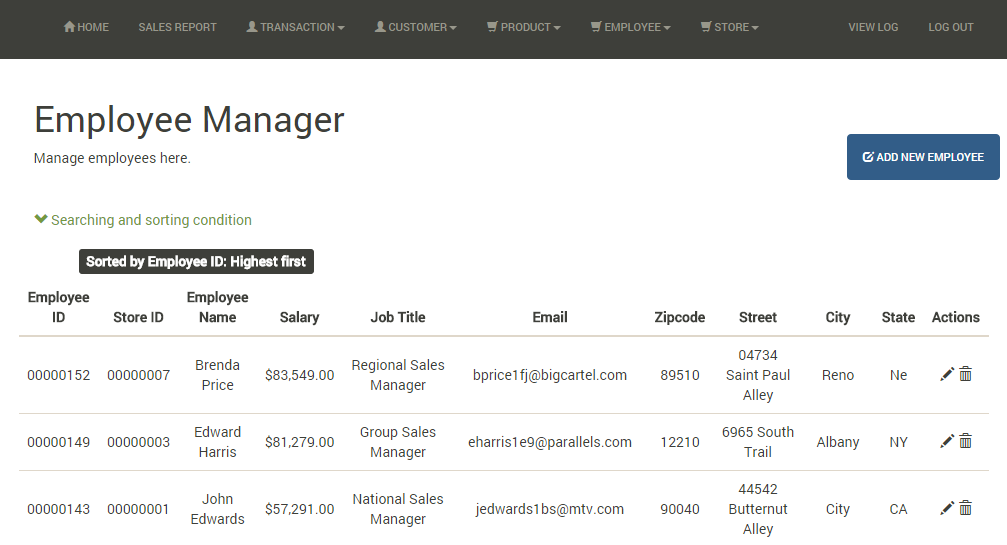


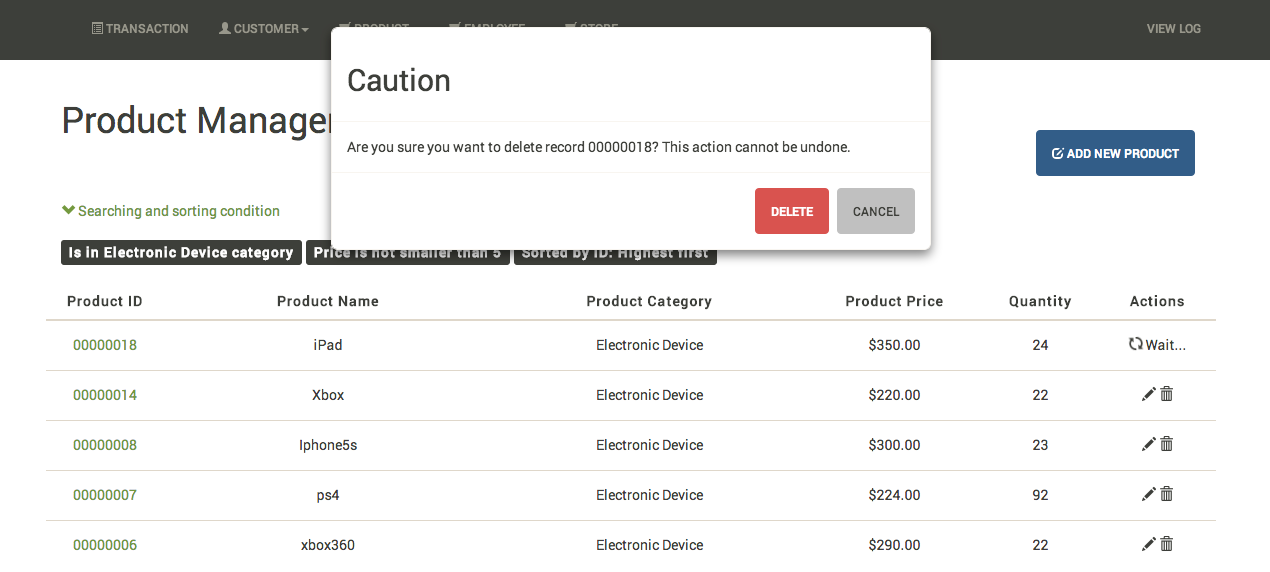
1. Searching and sorting panel gives detailed options for users



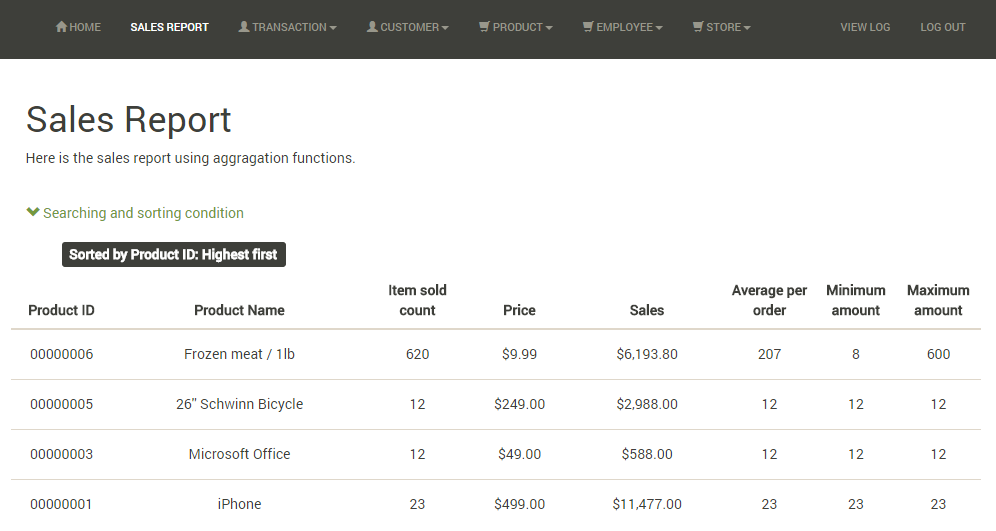
1. Tags – give users an overview of current searching and sorting conditions



1. In employers’ interface, user can add/sell product by pressing the button and fill the blanks.
2. In administrator’s mode, user can add, edit or delete most of the data. Other manager are similar to the employee manager
3. Delete product – a confirmation helps prevent careless operations.



1. Sales Report – accessible to all the users, using aggregation query to calculate sum, average, min, max values by giving conditions.



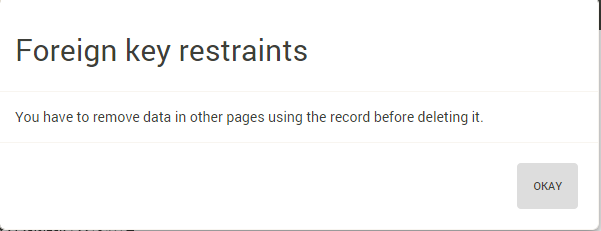
# Testing efforts

The system we designed is quite modular. We want to make the code readable and maintainable, so we separated the project into separated components: JSON schema, a single web page, and PHP services. It costs time making an automatic webpage and SQL statement generating system, but it prevented us from testing and debugging for small errors repeatedly.

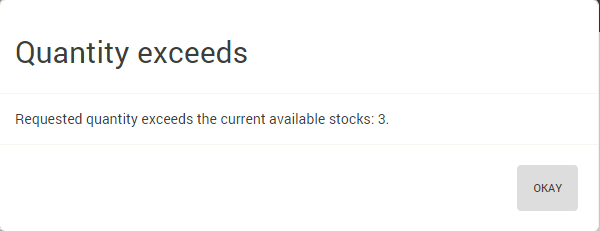
As illustrated in the graph, what we really need to do to optimize the system is to adjust the schema, and by doing this we minimized the risk of making bugs when programming. However, the design lacks a little flexibility. When we want to change a column slightly, we have to build a new auto-processing procedure for it.

There are some exceptions handled by the system, the others will be logged and the administrator can check the error and related operation easily.

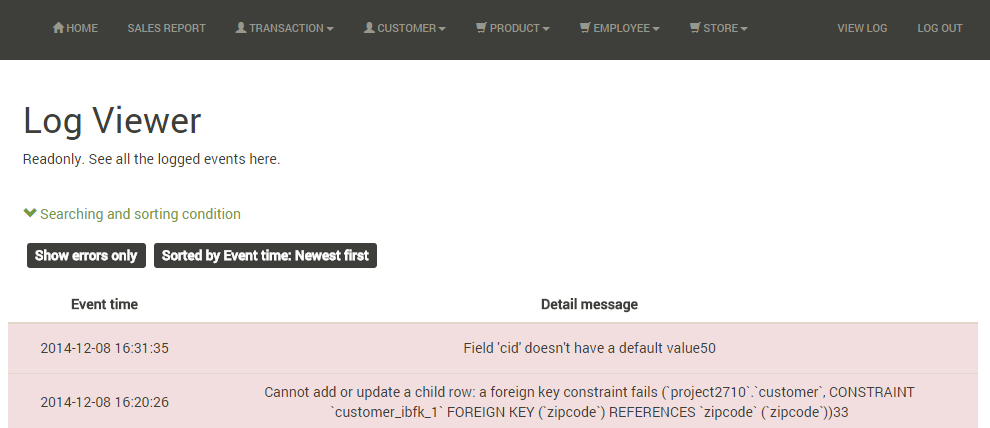
1. Foreign key constraint when deleting



1. Stock availability checking: a selling transaction won’t be allowed when requested quantity exceeds the current available stocks



1. Log viewer – other errors will be logged here



# Limitations and improvements

As mentioned in our testing and refining part, the system is not good at doing some complicated, especially nested queries. For safety reason we don’t allow user to execute SQL statements, so generating a flexible query by sending basic parameters seems is not that easy. We may want to use JSON or XML to implement more difficult queries in the future.

Another limitation of the project is that although there is a log viewer, we didn’t implement the undo or rollback feature. Currently all the exceptions should be handled manually by an administrator. We hope that we can make some of them automatic by implementing crash recovery.

Concurrency control is something we haven’t implemented too. The editing and adding mode are not exclusive, which means people could overwrite others’ ongoing work. It’s a new concept we just learned, and we are expecting to make the project more concurrent friendly eventually.