System Maintenance

Thomas Moffat, 51337, 4042

March 20, 2016

0.1 System Overview

This is a system designed for a small swimming club to allow them to remove the current paper based system for ordering kit and instead go entirely paper-less. This system allows customers to place orders using a web form which is then passed into a database, and then allows the Kit Coordinator to view all of these entries, as well as generate the order form for the suppliers. The system retains the customer details for future reference, if people need to know what they ordered previously.

A new customer can be created by the customer accessing the order form and filling in their details (string/int) which is then passed to a MySQL database. A new item order is created by them submitting their details and then redirecting to the next page, filling in the order form there and submitting it. This is again passed to the MySQL database. A new connection is created by calling the getConnection function in the Java program, which gets the connection details from kitorder.xml, and then Driver_Manager to create a connection to the database. The search function in the Java program gets the search string (string) typed in from the MainMenu and the order (string) and inputs these into an SQL statement with wildcards, returning the ResultSet as part of a dynamic JavaFX TableView. This is also how the TableView function works, just lacking the search string. To generate an order form, a Python script is called from the Java code, which then connects to the database, retrieves the relevant details for each item as a ResultSet and then inputs them into the spreadsheet using openpyxl, which can then be sent to the printers. In this, each item has its own sheet for clarity. The final non-trivial process in this program is the database dump feature, which makes a backup of the contents of the database in CSV files, then calls Mybatis, which executes a SQL script, dropping all of the tables in the database and then recreating them as new blank tables. Following this it calls a CSVReader function which reads a CSV file to recreate the items table of the database, as this never needs to change.

This program is connected to a MySQL database. The Customer, Order and Items are stored in separate, related tables which use foreign keys to ensure accuracy of data. When a new order is created, an appropriate SQL statement is generated and the order is inserted into the database.

0.2 Algorithms

Due to the nature of the code, for reasons explained Section 2.6 of the main report, there are no data manipulation algorithms in this report, although there are four quite complex SQL statements that are used in the program. They are given as follows, and discussed:

```
select o.ID, o.Orders, o.CustomerID, c.Name, c.
    Email_Address, c.Squad, o.OrderSize, o.OrderNumber
    , o.PaymentMethod, i.Item from INNER JOIN Items i ON i.
    idItems=o.Order where o.ID like ? or o.CustomerID like ? or
    c.Name like ? or c.Email_Address like ? or c.Squad like ?
    or o.Orders like ? or o.OrderSize like ? or o.OrderNumber
    like ? or o.NameOnGarment like ? or o.PaymentMethod like ?
    or i.Item like ? ORDER BY ?;
```

This is the search algorithm for the DBSearch.java function. This is set up as a prepared statement so that a SQL injection is impossible to undertake. Each of the question marks in this case rep-

resents the same thing, the search string. In the Java code this has % appended to the beginning and end to act as wild cards. The source code for this is as follows:

```
1                switch (ordering){
2                    case "Name A-Z":     order = "Name ASC";
3                                         break;
4                    case "Name Z-A":     order = "Name DESC";
5                                         break;
6                    case "Item":         order = "Order";
7                                         break;
8                    case "Order ID":     order = "ID";
9                                         break;
10                   case "Squad":        order = "Squad";
11                                        break;
12                   case "Customer ID:":order = "CustomerID";
13                                        break;
14                   case "Payment Method":  order = "PaymentMethod";
15                       break;
16                   default:             order = "ID";
17               }
18               PreparedStatement statement = c.prepareStatement("select o.
                     ID,  o" +
19                       ".CustomerID, c.Name, c.Email_Address, c.Squad, o.
                           Orders," +
20                       " o.OrderSize, o.OrderNumber, o.NameOnGarment, o.
                           PaidFor, " +
21                       "o.PaymentMethod, i.Item from Orders o INNER JOIN "
                            +
22                       "Customers c ON o.CustomerID = c.ID INNER JOIN
                           Items i ON" +
23                       " i.idItems=o.Orders where o.ID like " +
24                       "? or o.CustomerID like ? or c.Name like ? or c" +
25                       ".Email_Address like ? or c.Squad like ? or o.
                           Orders like" +
26                       " ? or o.OrderSize like ? or o.OrderNumber like ? 
                           or  o" +
27                       ".NameOnGarment like ? or o.PaymentMethod like ? or
                           i" +
28                       ".Item  like ? ORDER BY "+ order +";");
29
30               statement.setString(1, "%" + token + "%");
31               statement.setString(2, "%" + token + "%");
32               statement.setString(3, "%" + token + "%");
33               statement.setString(4, "%" + token + "%");
34               statement.setString(5, "%" + token + "%");
35               statement.setString(6, "%" + token + "%");
```

```
36                statement.setString(7, "%" + token + "%");
37                statement.setString(8, "%" + token + "%");
38                statement.setString(9, "%" + token + "%");
39                statement.setString(10, "%" + token + "%");
40                statement.setString(11, "%" + token + "%");
```

This code also shows how the program deals with setting the ordering, i.e. it takes a variable from the MainMenu and uses a switch case to convert it into a form that MySQL will understand. However the important bit is lines 30-40, where the input is mapped to each of the question marks in the SQL statement in order. This allows the user to put in anything, even a SQL statement and absolutely nothing will happen, as the special characters in the statement are being ignored and so are inserted into the database verbatim.

The next SQL statement is very similar to the previous one, it just doesn't have any of the prepared statement as it doesn't take directly user-editable inputs. This means that it takes a value selectable from a drop-down menu and uses that to order the results, but the user can't put their own string in its place. It goes thusly:

```
select o.ID, o.CustomerID, c.Name, c.Email_Address, c.Squad, o.
    Orders, o.OrderSize, o.OrderNumber, o.NameOnGarment, o.
    PaidFor, o.PaymentMethod, i.Item from Orders o INNER JOIN
    Customers c ON o.CustomerID = c.ID INNER JOIN Items i ON i.
    idItems=o.Order ORDER BY ?;
```

The code for how this part of the program deals with ordering is exactly the same as the previous one.

The third SQL statement is called when the database is remade. This dumps the contents of the Customers and Orders tables into two CSV files which can then be used for further reference or to repopulate the database after it has been wiped clean. It follows as such:

```
SELECT * FROM Customers INTO OUTFILE /Users/tsmoffat/
    kitordersystem/kitordersystem/kitordersystem/Customers.csv
    FIELDS ENCLOSED BY '"' TERMINATED BY ';' ESCAPED BY '"'
    LINES TERMINATED BY '\r\n'; SELECT * FROM Orders INTO
    OUTFILE /Users/tsmoffat/kitordersystem/kitordersystem/
    kitordersystem/Orders.csv FIELDS ENCLOSED BY '"' TERMINATED
    BY ';' ESCAPED BY '"' LINES TERMINATED BY '\r\n';
```

This statement is actually two almost identical statements acting one after the other. First everything is taken from the Customers table and put into a CSV file where each item is enclosed with "", each line is finished with a semi-colon and then a carriage return is performed to input the next row of data. This is exactly the same with the second statement except that takes all the values from the orders table.

The final complex SQL statement is actually a script, which is called when the database is reset. This drops every table in the database then remakes them blank. It is given here:

```
1 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
2 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
3 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES
    ';
```

```
SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT;
SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS;
SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION;
SET NAMES utf8;
SET @OLD_TIME_ZONE=@@TIME_ZONE;
SET TIME_ZONE='+00:00';
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO';
SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0;
CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8;
USE `mydb`;
DROP TABLE IF EXISTS `mydb`.`Customers`;
CREATE TABLE IF NOT EXISTS `mydb`.`Customers` (
  `ID` INT NOT NULL AUTO_INCREMENT,
  `Name` VARCHAR(45) NOT NULL,
  `Email_Address` VARCHAR(45) NOT NULL,
  `Squad` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE INDEX `ID_UNIQUE` (`ID` ASC))
ENGINE = InnoDB;
DROP TABLE IF EXISTS `mydb`.`Items`;
CREATE TABLE IF NOT EXISTS `mydb`.`Items` (
  `idItems` INT NOT NULL AUTO_INCREMENT,
  `Item` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idItems`),
  UNIQUE INDEX `idItems_UNIQUE` (`idItems` ASC))
ENGINE = InnoDB;
DROP TABLE IF EXISTS `mydb`.`Orders` ;
CREATE TABLE IF NOT EXISTS `mydb`.`Orders` (
  `ID` INT NOT NULL AUTO_INCREMENT,
  `Orders` INT NOT NULL,
  `OrderSize` VARCHAR(45) NOT NULL,
  `OrderNumber` VARCHAR(45) NOT NULL,
  `CustomerID` INT NULL,
  `NameOnGarment` VARCHAR(45) NULL,
  `PaidFor` TINYINT(1) NOT NULL,
  `PaymentMethod` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`ID`),
  INDEX `Order1_idx` (`Orders` ASC),
  INDEX `CustomerID_idx` (`CustomerID` ASC),
  CONSTRAINT `Order1`
    FOREIGN KEY (`Orders`)
    REFERENCES `mydb`.`Items` (`idItems`)
    ON DELETE NO ACTION
```

```
49      ON UPDATE NO ACTION,
50    CONSTRAINT `CustomerID`
51      FOREIGN KEY (`CustomerID`)
52      REFERENCES `mydb`.`Customers` (`ID`)
53      ON DELETE NO ACTION
54      ON UPDATE CASCADE)
55  ENGINE = InnoDB;
56  SET SQL_MODE=@OLD_SQL_MODE;
57  SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
58  SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

This is obviously a very long series of statements but