Sales Order and invoicing – desktop program

Jian Zhao
Dongfan Zhang

Background

- From existing sales orders stored in database, enable user to issue the invoices for chosen sales orders.

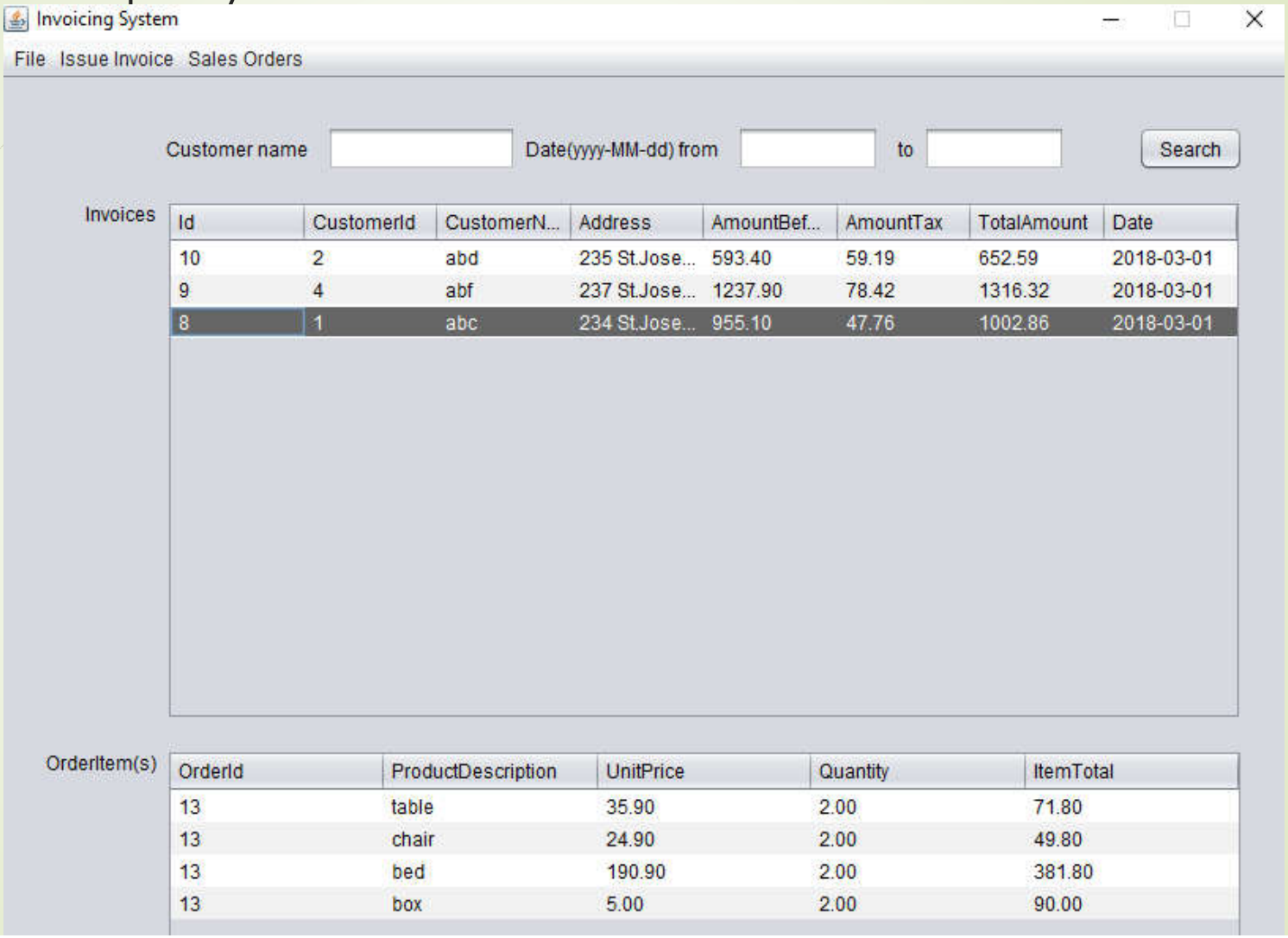
- Once invoice is issued, it will be stored into database, and corresponding order status will be changed to complete.

Solution overview

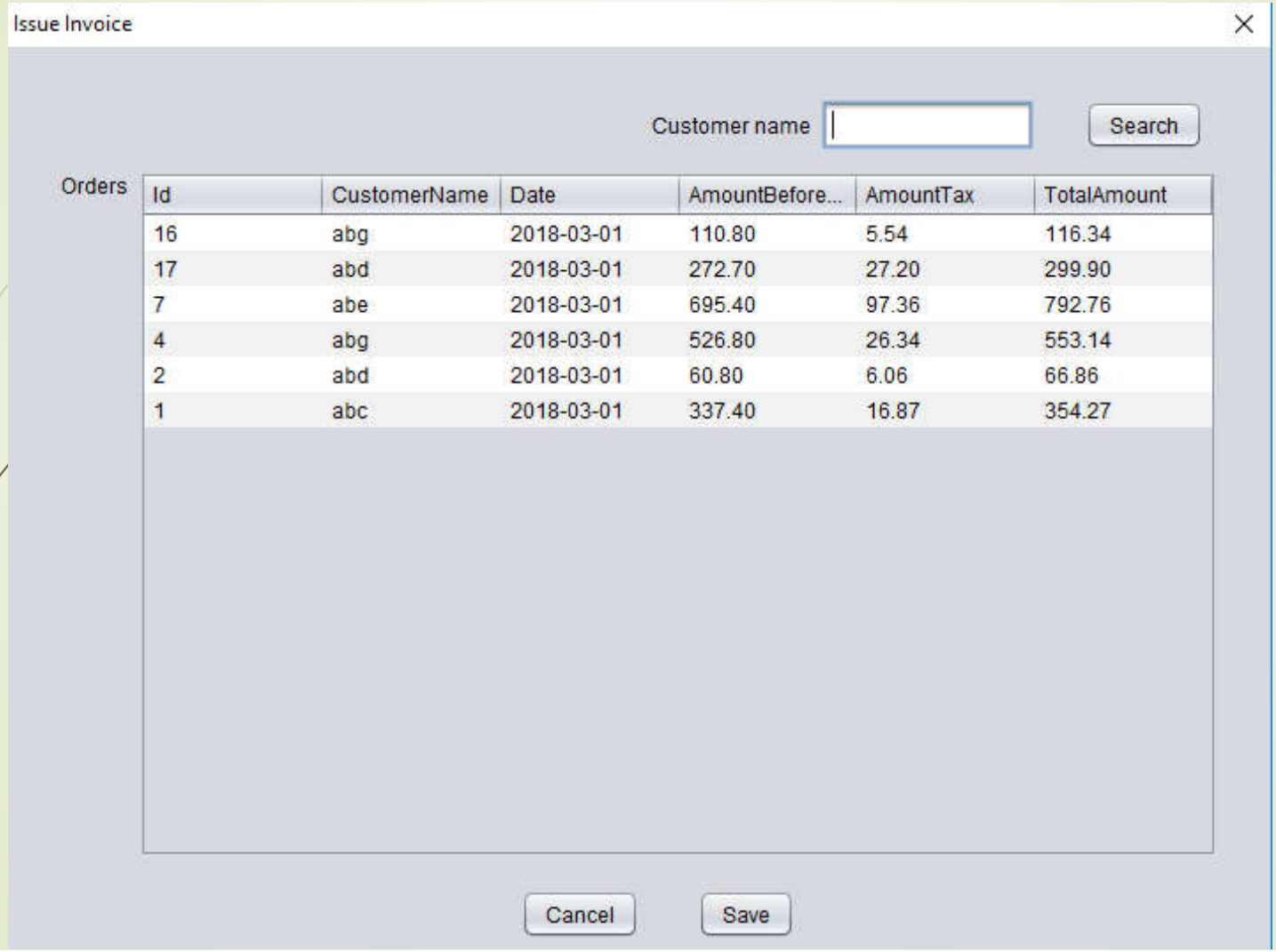
Functions to be implemented into program

- Query Sales orders
- Query Invoices
- Issue invoices
- export invoice to csv.
- export invoice to PDF.
- Export invoice and email as attachment

Invoice query

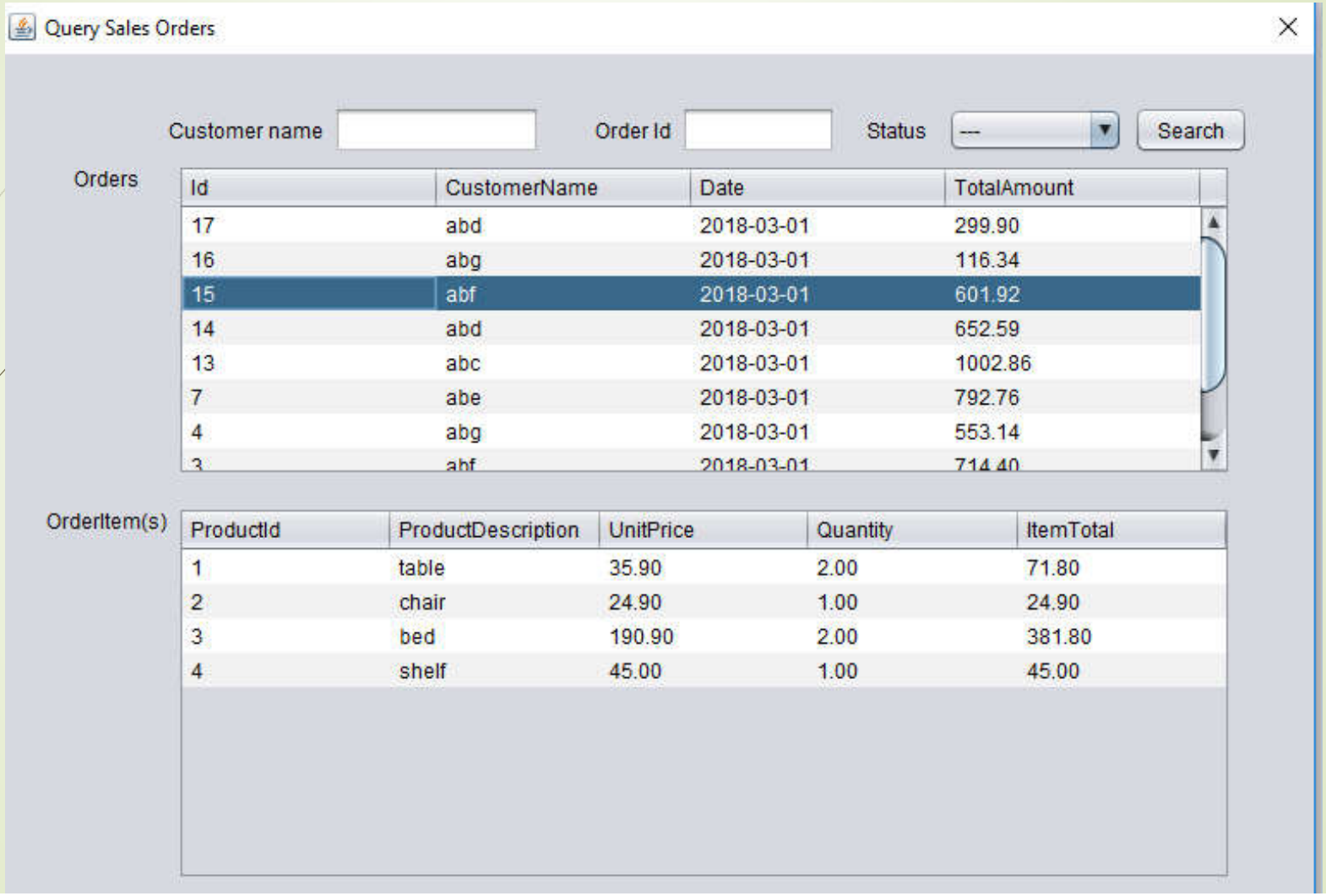


Issue invoices



Id	CustomerName	Date	AmountBefore...	AmountTax	TotalAmount
16	abg	2018-03-01	110.80	5.54	116.34
17	abd	2018-03-01	272.70	27.20	299.90
7	abe	2018-03-01	695.40	97.36	792.76
4	abg	2018-03-01	526.80	26.34	553.14
2	abd	2018-03-01	60.80	6.06	66.86
1	abc	2018-03-01	337.40	16.87	354.27

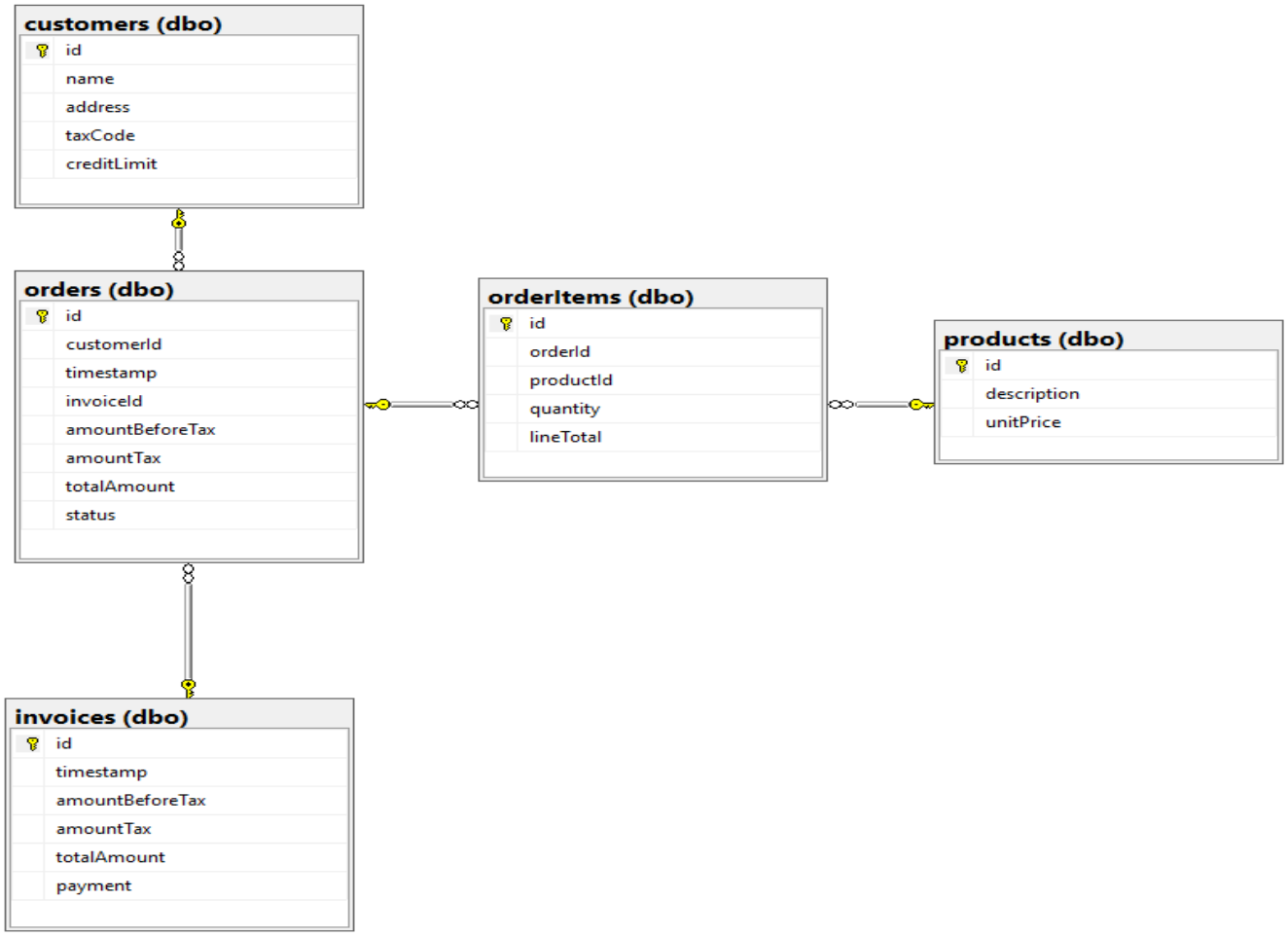
Sales orders query



Challenges and solution

- Database design and data integrity (proper foreign key, constraint, and implement triggers)
- Mapping the database designs (composition/ aggregation)
- Write the DAO (database class)according to program requirements for the various data queries (overload method, StringBulider to write queries)
- Manipulate data (work with Collections)
- Find out how to export invoice as PDF (library)
- Find out how to send emails via java (library)

Database design – 5 tables



Every table in database is turned into one java entity class with their own attributes/ fields plus some composition/aggregation from other classes.

- Public class Invoice {
- Private int id;
- private java.sql.Date timestamp;
- private BigDecimal AmountBeforeTax;
- private BigDecimal amountTax;
- private BigDecimal totalAmount;
- private BigDecimal payment;
- private List <SalesOrder> salesOrder;
- private Customer customer;

- public class SalesOrder {
- private int id;
- private int customerId;
- private java.sql.Date timestamp;
- private BigDecimal amountBeforeTax;
- private BigDecimal amountTax;
- private BigDecimal totalAmount;
- private OrderStatus status;
- private List<OrderItem> items;
- private Invoice invoice;
- private Customer customer;

DAO (DATABASE class)

Using database connection transaction ensure the data integrity for issuing invoice (insert record into invoices table, meanwhile modify status to "complete" for orders through update orders table)

Overload getOrders method for adapting different search functionality

Using StringBulider instead of String for concatenation sql statements, which are easier readable and more elegant

Working with Collections

- In our case, one invoice has one or several sales orders and one sales order can have multiple items. This often results to manipulate data collections.
- Sample code as below:
- List<Invoice> invoices = db.getInvoices(txtCustomerName.getText(), dateFrom, dateTo);
- for (Invoice invoice : invoices) {
- modelInvoices.addRow(new Object[]{
- invoice.getId(),
- invoice.getCustomer().getId(), invoice.getCustomer().getName(), invoice.getCustomer().getAddress(),
- invoice.getAmountBeforeTax(), invoice.getAmountTax(), invoice.getTotalAmount(), df.format(invoice.getTimestamp())
- });
- }

Export to PDF

- Using open source iText library for java
- Using rectangle to implement block control for positioning paragraph text
- Using table to display order items of invoices

Email with attachment using gmail

- Using library activation.jar and javax.mail.jar, we can realise the function of sending email with attachment using Gmail server.
- When user click export and send email. Program will first save csv to predefined folder then fetch the email address from database through customers table.
- We write a separate email class with send method to wrap up all the code for email, in the main code class we just need to pass the arguments to the send method to realize the function

```
private void miExCSVEmailActionPerformed(java.awt.event.ActionEvent evt) {
        exportCSV();
        String email = "";  String fileName = "";
String filePath = new File("").getAbsolutePath() + "\\export_invoices\\csv\\";
        final String user = "zdfmontreal13@gmail.com";
        final String password = "zoe20178";
        try {
            Object invoiceId = modelInvoices.getValueAt(jtInvoices.getSelectedRow(), 0);
            Invoice invoice = db.getInvoiceById(Integer.parseInt(invoiceId.toString()));
            email = invoice.getCustomer().getEmail();
            fileName = "invoice_" + invoice.getCustomer().getName() + "_" +
invoiceId.toString() + ".csv";
        } catch (SQLException ex) {
            Logger.getLogger(MainFrameInvoice.class.getName()).log(Level.SEVERE, null, ex);
}
        System.out.println(email);
        MailWithAttachment.send(user, password, email, "invoice", "please check invoice",
filePath + fileName, fileName);
    }
```

Future improvements

- Add user log in interface when the program starts
- Add different users roles like order entry, invoice issue, etc.
- Add Sales orders management (create new order, update order, delete order)
- Add payment function

Summary

By using various techniques, we successfully implemented planned functions for our Sales order invoicing mini program