CSCI 3901 Winter 2021

Assignment 5

Due date: 23:59 AST Friday, March 26th, 2021 in Brightspace

Problem 1

Goal

Access SQL through Java, practice developing SQL queries, and gain some exposure to XML.

Background

You work for the Northwind food distribution company. Management periodically wants a summary of the company's operation **over a period of time**.

Problem

Your job is to extract the summary information from the database, given a particular time period. You will store the summary information in a file that follows an XML format. Someone else will then use XML tools (notably XSLT) to convert your information into something that management will review.

There is **no specific class structure** for this problem. You are tasked with writing a command-line program with no constraints on the structure except for

- the program inputs, summarized below,
- the program outputs, summarized below, and
- **your code must compile**

Input

Your program will obtain the following information from the keyboard in the following order:

- The starting date for the period to summarize
- The ending date for the period to summarize
- The name of the file for the output

All dates will be in a `YYYY-MM-DD` format. For instance, valid text input for the date will be `2002-02-17`.

Output

Your program will **write all of its output to the specified file**.

You will extract and report data in 3 categories:

1. Customer information
    - Report the **customer name**, **address**, **number of orders in this period**, and **total dollar value of their orders in this period**.
2. Product information
    - Report, **for each product category**, the **category name** and **for each product in the category**, report the **name**, **supplier**, **units sold**, and **total dollar value of product sold in this period**.
3. Supplier information
    - Report, **for each supplier with products sold in this period**, the **supplier name**, **address**, **number of products sold**, and the **total dollar value of business that we sold from this supplier's products in this period**.

In all of the reporting, do not report any customers, products, or suppliers who have not had any interaction over the reporting period.

Your output file will be in an XML format. XML uses a set of tags to surround data to let you know what the data is. Some tags can be nested in other tags. HTML follows an XML-style format.

We will use a simple version of XML. The first line of your XML file should provide information on the version of XML to use. The following line will be sufficient:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

Following this first line, we get a set of nested tags to store the data. The starting tag has the format `<...>` and the matching ending tag has the format `</...>` (differing by the ending slash) where `...` is the tag name. The outermost tag is `period_summary`

Here is a description of the XML schema we will use in Document Type Definition (DTD) format (see https://en.wikipedia.org/wiki/Document_type_definition#XML_DTD_schema_example).

```
<!ELEMENT period_summary (period, customer_list, product_list, supplier_list)>
<!ELEMENT period (start_date, end_date)>
<!ELEMENT customer_list (customer*)>
<!ELEMENT customer (customer_name, address, num_orders, order_value)>
<!ELEMENT address (street_address, city, region, postal_code, country)>
<!ELEMENT product_list (category*)>
<!ELEMENT category (category_name, product*)>
<!ELEMENT product (product_name, supplier_name, units_sold, sale_value)>
<!ELEMENT supplier_list (supplier)>
<!ELEMENT supplier (supplier_name, address, num_products, product_value)>

<!ELEMENT start_date (\#PCDATA)>
<!ELEMENT end_date (\#PCDATA)>
<!ELEMENT customer_name (\#PCDATA)>
<!ELEMENT num_orders (\#PCDATA)>
<!ELEMENT order_value (\#PCDATA)>
<!ELEMENT street_address (\#PCDATA)>
<!ELEMENT city (\#PCDATA)>
```

```
<!ELEMENT region (\#PCDATA)>
<!ELEMENT postal_code (\#PCDATA)>
<!ELEMENT country (\#PCDATA)>
<!ELEMENT category_name (\#PCDATA)>
<!ELEMENT product_name (\#PCDATA)>
<!ELEMENT supplier_name (\#PCDATA)>
<!ELEMENT units_sold (\#PCDATA)>
<!ELEMENT sale_value (\#PCDATA)>
<!ELEMENT address (\#PCDATA)>
<!ELEMENT num_products (\#PCDATA)>
<!ELEMENT product_value (\#PCDATA)>
```

This means that a tag `period_summary` must contain nested tags for each of

- `period`,
- `customer_list`,
- `product_list`, and
- `supplier_list`.

In the tag `period`, the nested tag `start_date` is defined further below as `#PCDATA` — this simply means that the `start_date` tag will **not** contain any nested data and will instead just be a string, such as `<start_date> 1996-01-30 </start_date>` The tag `customer_list` will contain **zero or more** nested `customer` tags, as indicated by the `*` after the `customer` tag in the `!ELEMENT` clause.

While spacing doesn't matter in an XML file, you should always use line breaks and tabs to make the XML file readable by a person.

Sample output

```
<?xml version="1.0" encoding="UTF-8" ?>
<period_summary>
	<period>
		<start_date> 1996-01-30 </start_date>
		<end_date> 1996-02-02 </end_date>
	</period>
	<customer_list>
		<customer>
			<customer_name> foo </customer_name>
			<address>
				<street_address> 123 Hemming Way </street_address>
				<city> Brandon </city>
				<region> Manitoba </region>
				<postal_code> P3J 4V2 </postal_code>
				<country> Canada </country>
			</address>
			<num_orders> 30 </num_orders>
			<order_value> 11425 </order_value>
		</customer>
	</customer_list>
	<product_list>
		<category>
			<category_name> games </category_name>
			<product>
```

```
                        <product_name> game1 </product_name >
                        <supplier_name> a_supplier </supplier_name>
                        <units_sold> 100 </units_sold>
                        <sale_value> 500 </sale_value>
                    </product>
                </category>
        </product_list>
        <supplier_list>
                <supplier>
                        <supplier_name> a_supplier </supplier_name>
                        <address>
                                <street_address> 456 Falcoln Ridge </street_address>
                                <city> Saskatoon </city>
                                <region> Saskatchewan </region>
                                <postal_code> Q3C 1T8 </postal_code>
                                <country> Canada </country>
                        </address>
                        <num_products> 5 </num_products>
                        <product_value> 1250 </product_value>
                </supplier>
        </supplier_list>
</period_summary>
```

Constraints

- You may use any data structure from the Java Collection Framework.
- Write your solution in Java. **The solution code must be your own**.
- Use the `mysql JDBC` connection for Java.
- If in doubt for testing, I will be running your program on `timberlea.cs.dal.ca`. Correct operation of your program shouldn't rely on any packages that aren't available on that system.

Notes

- Use SQL vs Java for processing data as you deem best.
- Be sure to document your approach and any resources that you use.
- Look at where the bulk of the marks are in the marking scheme to help focus your efforts.
- You can run your queries against the `csci3901` database on `db.cs.dal.ca`. I will also make the `sql` file for the database available to you so that you can create your own copy of the database.

What to submit

- Your Java code
- A `.pdf` file containing an argument as to why your solution is ready to be deployed

Marking scheme (out of 25)

- Documentation - 3 marks
- Program design, organization, style – 2 marks
- Proper XML format, including indentation for readability – 3 marks
- Correct extraction of customer information – 4 marks
- Correct extraction of product information – 4 marks
- Correct extraction of supplier information – 4 marks
- Extensibility of your code (i.e. how easy it would be to report additional information for customers, products, or suppliers without copying code segments) - 2 marks
- Whether your argument that your code is ready to be deployed is convincing – 3 marks

  0 Argument is unconvincing

  1 Argument is somewhat convincing but it needs additional testing before deployment

  2 Argument is convincing enough to deploy in parallel with manual duplication of effort

  3 Argument is completely convincing, enough to deploy fully without manual duplication