CS5421:XML Project Report

Question 1

(a) Here, we use the `xmlroot()`, `xmlelement()`, `xmlagg()` and `xmlforest()` to export a xml-type data form postgre database. The code is show below.

```
SELECT xmlroot(
    xmlelement(
        name warehouses,
        xmlagg(
            xmlelement(name warehouse,
            xmlforest(
                warehouse.w_id AS id,
                warehouse.w_name AS name,
                xmlforest(
                    warehouse.w_street AS street,
                    warehouse.w_city AS city,
                    warehouse.w_country AS country) AS address,
                (SELECT xmlagg(
                    xmlelement(name item,
                    xmlforest(
                    item.i_id AS id,
                    item.i_im_id AS im_id,
                    item.i_name AS name,
                    item.i_price AS price,
                    stock.s_qty AS qty)
                    ) ORDER BY item.i_id)
                FROM stock, item
                WHERE stock.w_id = warehouse.w_id
                AND item.i_id = stock.i_id) AS items)
            ) ORDER BY warehouse.w_id)),
    version 1.0, standalone yes)
FROM warehouse;
```

(b) The XML document is lossless. Losslessness means that we can reconstruct the database using the XML document. For the warehouse table, we can get `w_id`, `w_name`, `w_street`, `w_city`, `w_country` from each warehouse element and the corresponding address element. For the stock table, we can `w_id`, `i_id` and `s_qty` from each warehouse element and corresponding item elements. For item table, we can get `i_id`, `i_im_id`, `i_name` and `i_price` from item elements in all the warehouse/items elements considering all the items appears in the warehouses. In a word, we can reconstruct all the three tables from the XML document. In conclusion, the XML document is lossless.

Question 2

In this question, we use DTD to describe the structure of the warehouses XML document. The root is `warehouses`. The `warehouses` has elements named `warehouse`. `Warehouse` has elements named `id`, `name`, `address`, `items`. The element `address` has elements name `street`, `city` and `country`. The `items` has elements names `item`. The element item has elements named `id`, `im_id`, `name`, `price` and `qty`. The DTD is shown below.

```
<?xml version="1.0" ?>
<!DOCTYPE warehouses [
  <!ELEMENT warehouses (warehouse+)>
  <!ELEMENT warehouse (id, name, address, items)>
  <!ELEMENT id (#PCDATA)>
```

```
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT address (street, city, country)>
  <!ELEMENT street (#PCDATA)>
  <!ELEMENT city (#PCDATA)>
  <!ELEMENT country (#PCDATA)>
  <!ELEMENT items (item+)>
  <!ELEMENT item (id, im_id, name, price, qty)>
  <!ELEMENT im_id (#PCDATA)>
  <!ELEMENT price (#PCDATA)>
  <!ELEMENT qty (#PCDATA)>
]>
```

Question 3

The key point of the three sub-questions is to come up with the corresponding XPATH for the elements which are asked for.

(a) For all the items that are available in the warehouse in quantity larger than 975 in Singapore, the XPATH is "`/warehouses/warehouse[address/country='Singapore']/items/item[qty>975]`". To satisfy the requirement of Question 3.(a), the XPATH need to be separated and inserted in corresponding positions. The code is shown below.

```
<?xml version="1.0"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
    <h1>Warehouses in Singapore</h1>
    <xsl:for-each select="/warehouses/warehouse[address/country = 'Singapore']">
      <h3>Warehouse name: <xsl:value-of select="name"/></h3>
      <table border="1">
        <tr>
          <td>Item Name</td>
          <td>Item quantity</td>
        </tr>
        <xsl:for-each select="items/item[qty>975]">
            <tr>
              <td><xsl:value-of select="name"/></td>
              <td><xsl:value-of select="qty"/></td>
            </tr>
        </xsl:for-each>
      </table>
    </xsl:for-each>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

(b) For all warehouses in Singapore or Malaysia, the XPATH is "`/warehouses/warehouse[address/country = 'Singapore' or address/country ='Malaysia']`". In the warehouse element, for the item with the largest qty, the XPATH is "`items/item[not(../item/qty > qty)]/name`". To satisfy the requirement of Question 3.(b), the XPATHs need to be inserted in corresponding positions. The code is shown below.

```
<?xml version="1.0"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
    <h1>Warehouses in Singapore</h1>
      <table border="1">
        <tr>
          <td>Warehouse Name</td>
          <td>Largest quantity item</td>
        </tr>
        <xsl:for-each select="/warehouses/warehouse[address/country = 'Singapore' or
address/country ='Malaysia']">
          <tr>
            <td><xsl:value-of select="name"/></td>
            <td><xsl:value-of select="items/item[not(../item/qty > qty)]/name"/></td>
          </tr>
        </xsl:for-each>
      </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

(c) For the sum of the quantities of the item 'Sunscreen' in Indonesia, the XPATH is "`sum(/warehouses/warehouse[address/country='Indonesia']/items/item[name ='Sunscreen']/qty/text())`". To satisfy the requirement of Question 3.(c), the XPATH need to be inserted in corresponding position. The code is shown below.

```
<?xml version="1.0"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
    <h1>Total quantity of Sunscreen in Indonesia</h1>
    <p><xsl:value-of select="sum(/warehouses/warehouse[address/country
='Indonesia']/items/item[name='Sunscreen']/qty/text())"/></p>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```