Search Technology for Media & Web

WiSe 2020/21

Programmieraufgabe 1

Abgabe: 18.12.2020, 16 Uhr MEZ

Diese Seite enthält (auf Englisch) die Aufgabenstellung der ersten Programmieraufgabe. Die Lösung der Aufgabe kann alleine oder als Team bestehend aus zwei Personen abgegeben werden. Falls zu zweit abgegeben wird, fügen Sie bitte der Einreichung eine Datei `partner.txt` bei, die den Namen des Partners enthält.
This page contains the description of the first assignment. The solution can be worked out individually or in a group consisting of two persons. Solutions by two people should also contain a file `partner.txt` with the partner's name.

Late Policy Es darf bis zu 5 Tage verspätet abgegeben werden. Die Maximalpunktzahl verringert sich pro verspäteten Tag um 5 Prozent (d.h. am Tag 5 kann nur noch maximal 75 Prozent der Gesamtpunktzahl erreicht werden).
The assignment can be handed in up to 5 days late. The maximal score is reduced by 5 percent for each day (thus on day 5 a maximum of 75 percent of the total score can be reached).

Virtual Machine (VM): we provide you with a VirtualBox disc image that runs Ubuntu Linux and has everything installed that is needed for the assignment. Your code must compile and run within this VM without any modifications to the VM (i.e., without installing any packages or libraries, etc.). You may use the `AD_assigment_1` directory provided. The username is `ad`, there is no password.
Click here to download image. **Warning: file size is 1.7GB**
To use the image, first unzip the file and then run VirtualBox. Under Machine select New, provide a name, choose type Linux and Ubuntu (32 bit). Choose a memory size (512MB should be OK), then select "Use an existing virtual hard drive file", and click on the folder-icon to select the disc image that you downloaded.

Assignment 1: Data Loading

The purpose of this assignment is to

1. design a database schema for a given dataset,
2. shred XML data into csv-files, and
3. create tables in a database and populate them via the csv-files.

The dataset consists of ebay auction data, given by a set of XML files. Each XML file is valid with respect to the given DTD. For Point (1) you will work out a convenient relational schema to represent the given auction XML data. This includes identifying the primary keys of each table, and explaining why the schema is "good", i.e., why it adheres to the desired normal forms of database design. Point (2) is the main part of this assignment. You are asked to write a Java program that reads in the XML files, and writes out csv-files, one for each table in your schema. In Point (3) you are asked to write SQL-scripts which create your tables in a MySQL database, and which load the csv-files into these tables. Finally, you are asked to test your database by running a few SQL-queries over it.

Note: this assignment is taken from UCLA's course "CS144 Web Applications". We gratefully appreciate Junghoo Cho's consent to use his material.

Part A: Download and examine the XML data files

First, download the XML-encoded ebay auction data set ebay-data.zip and unzip the file in the `$EBAY_DATA` directory (`/home/ad/ebay-data`), in our VM, in a terminal (press CTRL-ALT-T), type the following:

```
$ wget http://user.informatik.uni-bremen.de/~smaneth/DST1/ebay-data.zip
$ unzip -d $EBAY_DATA ebay-data.zip
$ rm ebay-data.zip
```

1. As a *small data set* of initial experiments and debugging we suggest to use just one file: `items-0.xml`. It contains 500 auctions, comprising about 900kb of plain-text data.
2. Your system must also work for the *large data set* consisting of all 40 XML files. There are about 20,000 auctions, comprising about 38mb of plain-text data.

Each XML is valid with respect to the DTD specified in the file `items.dtd`. Your first task is to examine the DTD and the XML files to completely understand the data you will be starting with. You will be translating this data into relations and load them into MySQL server

on our VM. Please read the auction data help file in `items.txt`. The provided data was captured at a single point in time; more precisely, on December 20th, 2001, one second after midnight. It contains items that have been auctioned off, and items that are "currently" up for auction.

Note that UserIDs of sellers and bidders uniquely identify a user. Whenever a user appears in the data, his/her Rating, Location, and Country information are the same. For more, read `items.dtd`.

Part B: Design your relational schema

Now that you understand the data you will be working with, design a good relational schema for this data. Iterating through the following three steps should help you in the design process.

1. List your relations. Please specify all keys that hold on each relation. You need not specify attribute types at this stage.
2. List all completely nontrivial functional dependencies that hold on each relation, excluding those that effectively specify keys.
3. Are all of your relations in Boyce-Codd Normal Form (BCNF)? If not, either redesign them and start over, or, explain why you feel it is advantageous to use non-BCNF relations.
4. Are all of your relations in Fourth Normal Form (4NF)? If not, either redesign them and start over, or, explain why you feel it is advantageous to use non-4NF relations.

Document these steps in the README.txt file for submission. Alternatively, a PDF file README.pdf would be acceptable as well.

Part C: Write a data conversion program

You will write a Java program that converts the XML data into csv-files that are consistent with the relational schema from Part B. For your csv-files you only need to make two decisions:

1. which delimiter to use (obviously you cannot use a comma, nor any other symbol that appears in the data!), and
2. which end-of-line character to use.

The MySQL LOAD command allows you to specify the delimiter (via `FIELDS TERMINATED BY`) and the end-of-line character (via `LINES TERMINATED BY`). Your Java program can use either DOM or SAX parsing. It is more challenging to do it via SAX and we encourage you to use SAX. An immediate advantage of using SAX, is that the resulting program uses **hardly any memory!** Thus, it can handle huge XML files, while the DOM solution can only handle sizes fitting into memory. To help you get started, we are providing

- a parser skeleton MyDOM.java. It reads the XML file given as argument into a DOM in-memory structure. It then recursively traverses through all element and text nodes. At an element node it prints its name, and all its attribute-value pairs. At a text node it prints the text content.
- a parser skeleton MySAX.java. It reads the XML file given as argument and parses it as a stream (i.e., it does **not** load it into memory!). It prints start and end-element events with the corresponding name, and prints out all attribute value pairs of an element, and text nodes (where some special characters such as CR and LF are printed as code).

On the VM, place these java files in the AD_Assignment_1 directory, compile with `javac file.java`, and run with `java file xmlFile.xml`, where xmlFile.xml is some XML file, e.g., `$EBAY_DATA/items-0.xml`.

Note: You are not permitted to use any java libraries or packages other than those that come with java and are available on the VM, e.g., you can use standard libraries such as java.util.* etc.

Notes on MySQL Datatypes

We note that some `Description`-elements in the XML data are quite long. If any text is longer than 4000 characters, then you must **truncate it at 4000**. We suggest you use the MySQL data type `VARCHAR(4000)` to store descriptions, and truncate descriptions that are too long to fit. For attributes representing dates/times, we suggest you MySQL's `TIMESTAMP` type; be careful to convert correctly into this date format. For attributes representing money, we suggest to use `DECIMAL(8,2)`.

Duplicate Elimination

When transforming the XML data to relational tuples, you may discover that you generate certain tuples multiple times, but only want to reatain one copy. To solve this, you must implement duplicate-elimination as part of your conversion program (e.g., via hashing). Do NOT use the MySQL bulk loader to eliminate duplicates, or, temporary tables and the "SELECT DISTINCT" feature. The csv-files generated by your Java program must contain correctly duplicate-eliminated tuples.

Notes on CR/LF issues

If your host OS is Windows, you need to pay particular attention to how each line of a text file ends. By convention, Windows uses a pair of CR (carriage return) and LF (line feed) characters. Unixes use only a LF character. Therefore, problems arise when you feed a text file generated from a Windows program to a Unix tool (such as `mysql`). If you encounter this problem, you may want to run the `dos2unix` command in the VM (type `dos2unix --help` for more information).

Part D: Load your data into MySQL

Now create and populate your tables in MySQL inside the `ad` database. You can first experiment interactively by executing `mysql` and then typing in SQL commands by hand. When you create your tables via the `CREATE TABLE` command, be sure to specify a `PRIMARY KEY` for those tables that do have a primary key! Use the MySQL LOAD command to load data from a csv-file into a table. A table `T` may be dropped via the `DROP TABLE T;` command; use `DELETE FROM T;` to delete all tuples from `T`.

Automating the Process

MySQL can read a sequence of commands from a file and execute them in batch mode. This is done via the `mysql < file` command. Write three such batch files, named `create.sql`, `drop.sql`, and `load.sql`. The `create.sql` file will look something like:

```
CREATE DATABASE IF NOT EXISTS ad;
USE ad;
CREATE TABLE item ( .... );
CREATE TABLE xyz (.... );
...
```

Similarly, the drop.sql should drop each of your tables **THAT EXISTS**, by a sequence of `DROP TABLE` commands. Lastly, the `load.sql` contains MySQL LOAD commands. The file will look something like:

```
LOAD DATA LOCAL INFILE 'item.csv' INTO TABLE item FIELDS TERMINATED BY ...
LOAD DATA LOCAL INFILE 'xyz.csv' INTO TABLE xyz FIELDS TERMINATED BY ...
...
```

To automate the entire process, create a *bash shell script*, name `runLoad.sh`. Your script should do the following things:

1. Drop any existing relevant table from the `ad` database **IF EXISTS**.
2. Create all relevant tables in the `ad` database.
3. Compile and run your XML-to-CSV converter to generate fresh csv-files.
4. Load the data into MySQL.
5. Delete all temporary files (including the csv-files).

Here is a sceleton of what your `runLoad.sh` script can look like:

```
#!/bin/bash

# Run the drop.sql batch file to drop existing tables.
# Inside the drop.sql, you should check whether the table exists. Drop them ONLY if they
exist.
mysql < drop.sql

# Run the create.sql batch file to create the database (if it does not exist) and the
tables.
mysql < create.sql

# Compile and run the convertor
javac MySAX
java ...

# Run the load.sql batch file to load the data
mysql ad < load.sql

# Remove all temporary files
rm ...
...
```

Part E: Test your MySQL database

The final step is to take your newly loaded database for a "test drive" by running a few SQL queries over it. As with your initial database creation, use MySQL to test your queries interactively, then set up a batch file `queries.sql`, that runs them in batch mode later. First, try some simple queries over just one table, then more complex queries involving joins and aggregation. Make sure the results look correct. When you are confident that everything is correct, place queries for the following specific tasks into the `queries.sql` file:

1. Find the number of users in the database.
2. Find the number of items in "New York", i.e., itmes whose location is *exactly* the string "New York". Pay special attention to case sensitivity. E.g., you should not match items in "new york".
3. Find the number of auctions belonging to exactly four categories. Be careful to remove duplicates, if you store them.
4. Find the ID(s) of current (unsold) auction(s) with the highest bid. *Remember that the data was captured at December 20th, 2001, one second after midnight. Pay special attention to the current auctions without any bid.*
5. Find the number of sellers whose rating is higher than 1000.
6. Find the number of users who are both sellers and bidders.
7. Find the number of categories that include at least one item with a bid of more than $100.

Thus, your queries.sql should contain seven queries of the form `SELECT ... FROM ..`, one for each of the above tasks, and exactly in the order as specified above.

Sample Output of some Count Queries

To help you check whether your program loads the corrent amount of entries into your tables, here are some numbers that you should obtain: the number of entries in the item-table is 19532. The number of entries in the table that associates items with categories is 90269. The number of sellers is 13129. The number of bidders is 7010. The number of items that have a buyPrice is 1959. The number of items that have longitude/latitude information is 13090.

Efficiency versus Correctness

We grade your code for correctness. Efficiency is not the prime goal. However, your code must run in a *reasonable* time. Even on the VM it should not take more than 2 minutes to complete `runLoad.sh`. We give a penalty of 20% if your code takes longer than two minutes (on a non-VM setup), and zero points if it takes more than five minutes. On the machine we will use to grade, our sample solution takes around 10-20 seconds.

What to submit:

You should submit a file **LAST_FIRST.zip** (where LAST is your last name, and FIRST is your first name) containing these **seven (or eight) files** (and no directories whatsoever):

- A plain text file **README.txt**, containing your relational schema design, answers to the three items listed in Part B, as well as anything else you would like to document. (note: a PDF file **README.pdf** would be acceptable as well)
- Your MySQL batch files **create.sql**, **drop.sql**, **load.sql**, and **queries.sql**.
- The shell script named **runLoad.sh** as described above.
- The Java file **MySAX.java** (or **MyDOM.java**) containing the source code of your converter.
- (optionally) **partner.txt** containing your partner's details.

Note that `queries.sql` is not executed from the shell script. Do **not** submit your VM image or hard drive file!!

Submission Instruction

To submit, upload your zip-file on Stud.IP in the folder that will be provided.