Deliverables

The deliverable for this homework is mainly a web site that operates as a front-end to a SQL database, also including the supporting files `ERModel.pdf`, `TennisSchema.sql`, `DataImporter.py`, and `DynamicCalculations.sql`.

Pair Programming

You are required to work with your assigned partner on this assignment. You must observe the pair programming guidelines outlined in the course syllabus — failure to do so will be considered a violation of the Davidson Honor Code. *Collaboration across teams is prohibited and at no point should you be in possession of any work that was completed by a person other than you or your partner.*

1 Introduction

In this assignment, we will obtain un-normalized data from Jeff Sackmann's ATP Tennis dataset, and we will build a website that performs queries and displays information. Our goal is to implement a start-to-finish database system, from data (un-normalized) to presentation (user-facing website).

1.1 Material

The material a set of CSV files that can be obtained from

`https://github.com/JeffSackmann/tennis_atp`

Download the whole set of files (using the download link, or using `git clone`), but focus at the `atp_matches_XXXX.csv`, where XXXX is the year from 1968 to 2020.

Each file contains, among other fields, a tournament name and date, surface type (Clay, Hard, etc), and information about the players who won or lost (name, hand, height in cm, country of origin, rank). Each match has a tournament, and date. There's also the score of the match in the "score" column. Please refer to the file `matches_data_dictionary.txt` for an explanation of the fields.

2 (20 pts) Model

We often need to model database schemas based on functional requirements specified by application users. However, application users typically describe their functional requirements very informally, and it is up to the database designer to eliminate any ambiguities in the system description as the model is created.

Deliverable 1. Create an Entity-Relationship model (in a PDF file called `ERModel.pdf`) of a database to manage the information about the tennis matches using SQL, based on the informal requirements below:

1. List all players, tournaments, matches. Identify reasonable attributes for players, tournaments, matches. In addition to the obvious arguments, matches should store the score, number of sets.

2. The statistics about winners and losers, provided in the CSV files, refer to players in the match. We would like to obtain aggregate statistics about a specific player, for example: "how many double faults player X had, in average, on matches won between 1971 and 1975". You should *not represent* this particular information, but your model should be able to *support* this kind of queries.

3. Aggregate statistics about rank, hand (leftie vs. non-leftie) should be supported. For example: "which percentage of matches in tournaments with draw size bigger than 64 were won by lefties?".

Note that the CSV file might induce you that information about the winner is an attribute of the match. It is not: a winner is a player, and that player should be identifiable, and aggregate statistics for that player for multiple matches should be attainable. Also, do not store information about a player's age – calculating a player's age dynamically upon first identifying him/her would be conceptually easy, and your time (as a student) can be used better. Do not forget to identify attributes and primary keys in the entities. Finally, there is no need to store information about seed entries.

The description above is certainly vague and even open-ended. I am precisely evaluating your ability to parse such kind of specification, using Entity-Relationship in order to produce a working model that fully supports the queries requested by users. Vagueness and open-endness are part of the problem. **Consulting the professor in office hours to check the model is allowed and encouraged.**

3 (15 pts) Schema and Integrity Constraints

Deliverable 2. You should create a database schema (in a SQL script `TennisSchema.sql`) that is a physical realization of your model above, respecting the following integrity constraints:

1. If a player is deleted, matches involving that player should not be deleted – appropriate entries should be set to NULL instead.
2. If a tournament is deleted, all matches within that tournament should be deleted.
3. The rankings should never be negative[1], information about dominant hand should be only 'L' or 'R', etc. Please provide integrity constraints that make sure that the data accepted by the database is always clean.

Make sure to identify all primary and foreign keys. Also, the IDs in the CSV files are not something we can reasonably trust (for example, see the description for `match_num` in the `matches_data_dictionary.txt` file. Create your own IDs for your entities. Make a web search for "AUTO INCREMENT" in SQL and be amazed.

4 (15 pts) Importing the dataset into MySQL

To import the dataset into MongoDB, you should read the information from the CSV files using your preferred language, although this should be more effectively done in a language like Python. Importantly, **there is no skeleton file.**

Deliverable 3. Create a program called `DataImporter` (if Python, `DataImporter.py`; if Java, `DataImporter.java`) that creates the tables and imports *all* the matches information between 1968 and 2020. Note that I'd like this program to insert the information in the database using **one pass**, that is, you should not read any line of the CSV file more than once.

5 (25 pts) Functions, Procedures, Triggers, Views

Deliverable 4. Create a file `DynamicCalculations.sql` containing:

1. A function **aceCount(name, start, finish)** that calculates the average number of aces, given a player name and a time frame between two dates. Note that you can

[1] Except your professor's tennis rank, which is -1.

compare dates in SQL using < and >.

2. A procedure **showAggregateStatistics(name, start, finish)** that yields a result set containing all aggregate statistics of a player during a specific time frame. The way you aggregate statistics is up to you: maybe you want to show the total number of aces, or the average number of aces per match, etc. Include the data on double faults, break points saved, etc, in your statistics.

3. A view **TopAces** that lists the top 10 players based on number of aces, across all matches.

4. A trigger **onInsertionPlayer** that, upon inserting players, replaces a reference to the country code 'RUS' (and another former country of the block, say 'EST') to 'USR', for the former USSR[2]. Note that you do not want to update the whole table for every insertion; you want to update just the entry that is being inserted.

Note that a variety of functions and procedures similar to the ones you coded could be easily conceived with similar code. You are implementing only one function and one procedure, but going from 2 to 20 would cost only time – not extra technical knowledge.

6 (25 pts) A Web Frontend with MongoDB and PHP

Your users would like to obtain statistics about players, tournaments, counties, etc, and they would like to query your database using a website. In this part of the homework, you will exercise:

1. Your ability to learn new tools and languages given only documentation;
2. Your sense of style in designing a front-end application[3]
3. Technically, you will see how websites, databases, and users inter-operate.

6.1 HTML Forms

You know that webpages are written using HTML. In HW2, you already worked with HTML, and in this assignment you additionally need to use HTML forms, PHP, and JavaScript. Lets start with HTML forms:

`https://www.w3schools.com/html/html_forms.asp`

[2]Maybe storing codes as they were *then* makes more sense; in any case, this is simply an exercise.
[3]If you are taking Software Design, this should be familiar.

Please read the whole section to learn about forms. In addition, you will need PHP and JavaScript.

6.2 PHP

HTML files are hosted in a *web server*, that fetches local files upon request, and transfers them to the other side of the connection, where they are interpreted by *browsers*. The web server can also execute *server-side* scripts contained in these HTML files as they read them, so the files obtain data dynamically from a local or remote database. The web server reads the server-side script, and amends the HTML file with dynamic contents, typically obtained from a database.

A common server-side scripting language is PHP, which is embedded in HTML files. Please refer to material here:

`https://www.w3schools.com/php/` (read the "PHP Tutorial" and "PHP Forms" sections)

In order to install PHP on macOS, run in the terminal:

```
brew update
brew install php
```

If `brew install php` fails because you already have PHP, issue a `brew upgrade php` instead.

On Windows, you need to download the software and perform some additional configuration. Here is a short video that documents this process for Windows users:

`https://www.youtube.com/watch?v=9uVZbVEtf8Q`

After installing, let's test your PHP installation. First, you should run a web server. Fortunately, PHP version 8 comes with a web server of its own, and this assignment comes with two PHP examples. For the first test, **in the same directory where your PHP files reside**, run

```
php -S localhost:3333
```

This will launch a web server at the local machine, port 3333. Then, using your browser, navigate to

`http://localhost:3333/index.html`

After reading the HTML forms tutorial, please inspect the `index.html` file. This file

will collect HTML form data, and will send it to the server, along with a request to obtain the `handler_page.php` webpage. The server will then *interpret* the PHP from the `handler_page.php` file, which captures the HTML form data and prints out a simple message. Database connections are not made in this first example – the next example will include a DB connection based on form data.

For the second example, you have **first** to setup your Activity 4 database and adjust username and password in the `handle_page_db.php`, if necessary.

Our second example is more interesting because it connects to MySQL to retrieve data. Using your browser, navigate to

`http://localhost:3333/index_db.html`

This file will collect a table name using HTML form data, and will send to the server, along with a request to obtain the `handler_page.php` webpage. Again, the server will interpret this PHP file, but now it will use the table name sent via HTTP post, and it will dump the contents of that table into the output. The output is crude because there is no format, but you can format the data using HTML elements, and rely on your CSS for presentation formatting.

Please inspect these files' syntax - they should be treated as setup documentation.

6.3 JavaScript

Once you get information from the server (item 2 of next section), the HTML generated remotely should be manipulated for better presentation. JavaScript is the language that runs inside browsers, allowing us to sort tables, search in tables, etc. Here are tutorial pointers for JavaScript:

`https://www.w3schools.com/js/default.asp` (read just the "JS Tutorial" part)
`https://www.w3schools.com/howto/howto_js_filter_lists.asp`
`https://www.w3schools.com/howto/howto_js_filter_table.asp`
`https://www.w3schools.com/howto/howto_js_sort_list.asp`
`https://www.w3schools.com/howto/howto_js_sort_table.asp`
`https://www.w3schools.com/howto/howto_js_filter_dropdown.asp`

Danger! The sorting examples above have two problems: (i) they implement sorting instead of using a library; (ii) implement sorting using bubblesort, a quadratic algorithm! Clearly, the tutorial writers are not computer scientists. Your sorting algorithm, a functional requirement

described below, should be a mergesort or a quicksort that runs in time O(nlog(n)).

6.4 Functional Requirements

Your web frontend should use forms and make available for the user the following queries:

1. Display all information from a player given the player's name. A player's name is specified via a text box.
2. Display the general attributes of tournaments, given a year. The selection of year is done with combo boxes. This means that the webpage that allows the users to select the year is generated by the webserver dynamically, using PHP. Once the data is loaded on the client, there are buttons that allow users to sort the data *locally*, using Javascript. Sort the tournaments by name, date, etc, **without making another query to the server**, that is, Javascript should be used to do that job by manipulating the HTML elements locally.
3. Display the aggregate statistics about a player between two dates. The dates are specified via combo boxes. The procedure **showAggregateStatistics(name, start, finish)** should be called with PHP in order to obtain the right result set.

At this point, you should have a sense of what is done in the client side (in the browser, with JavaScript), and in the server side (in the webserver, with PHP). The only thing missing here is that web developers use many prebuilt libraries for tasks like sorting and filtering data with JavaScript. You are not doing it here, but if you would like me to give a demonstration of this in class, let me know.

Good luck,
- Hammurabi