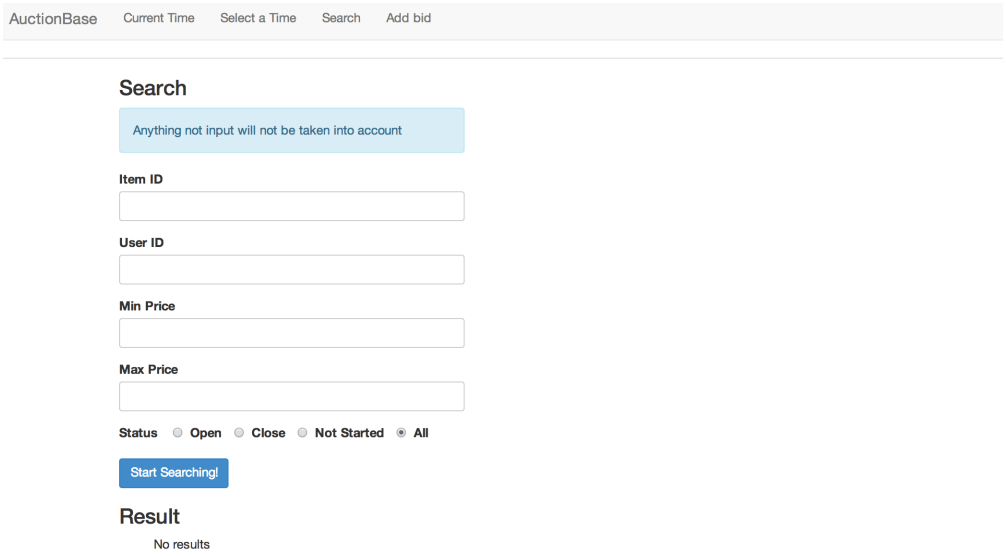
CS 564: Database Management Systems

University of Wisconsin - Madison, Spring 19

AuctionBase Project: Database and the Web

Overview

As a baseline, you will design a set of queries and updates for your AuctionBase system and create a simple web interface for them using the Python web.py framework. Before you get started, we recommend that you **read through the** Getting Started with web.py and Jinja2 **supporting document in its entirety!**



Task A: Getting started

- **Step 1:** *Download auctionbase.zip and unzip it.* This directory contains three subdirectories `create_auctionbase`, `ebay_data`, and `web.py`. Directory `create_auctionbase` contains the necessary files to parse the Ebay data and generate the backend for AuctionBase. The provided files correspond to an extended solution to project 1.
  In directory `web.py`, you should see the following files and directories:
  - `auctionbase.py` – Your "main" application. Responsible for handling requests from the browser.
  - `sqlitedb.py` – Your database manager. Responsible for interacting with your database.

- `templates/` – Directory that contains your template files that correspond to your various URLs. Every web page will require a new template file in this directory.
- `lib/` – Directory that contains the library files for web.py and Jinja2. **Warning: Do not modify any of the files in the `lib/` directory!**

- **Step 2:** *Generate your SQLite Database*
  Go to directory `create_auctionbase`. Use the `runParser.sh` script to extract the Ebay data. Also, use the `createDatabase.sh` script to generate the AuctionBase SQLite database (`.db`) binary file and move it into the `web.py` directory.

- **Step 3:** *Familiarize yourself with web.py and Jinja2*
  If you haven't already, go ahead and read through all of the *Getting Started with web.py and Jinja2* supporting document – this should give you a detailed guide of the starter code, and will demonstrate how to complete the required functionality for the assignment. (We **strongly** recommend that you do this, especially if this is your first time building a web application in Python!)

Task B: Data Integrity — Reading Only

This part requires that you only familiarize yourselves with the integrity constraints in the provided AuctionBase backend implementation. **You do not have to implement anything for this part**.

Task B.1: Adding Support for Current Time

The original auction data that we provided for you in JSON, which you translated into relations and loaded into your AuctionBase database in Part 1, represents a single point in time, specifically, one second after midnight on December 20th, 2001 (represented in SQLite as 2001-12-20 00:00:01).

To support full functionality and to simulate the true operation of an online auction system in which auctions close as time passes, we extended AuctionBase to maintain a fictitious "current time" in the database. To this end, we added a new one-attribute table to the AuctionBase schema that represents this current time.

(**Warning**: Do not try to call the attribute in your table `current_time` – it turns out that's a reserved keyword in SQLite.)

This table should at all times contain a single row (i.e., a single value) representing the current time of your AuctionBase system. **In this part of the project, when we ask you to simulate time advancing in AuctionBase, you'll do so by updating this table.**

For starters, the table is initialized to match the single point in time we've previously mentioned: 2001-12-20 00:00:01. We have also extended `create.sql` to create this table. **Do not edit or delete anything related to current time from this file.**

Task B.2: Adding Constraints and Triggers to AuctionBase

Before reading through this part, please read the **Referential Integrity in SQLite** support document. If you find the material in the optional Constraints and Triggers activity insufficient, you may also want to refer to the SQLite documentation for the `CREATE TRIGGER` and `DROP TRIGGER` statements. Finally, you can also refer to the documentation on `PRIMARY KEY`, `UNIQUE`, and `REFERENCES` declarations in the SQLite `CREATE TABLE` statement documentation. Be aware that SQLite constraints and triggers do not conform exactly to the SQL-99 (SQL2) standard.

If the data in your AuctionBase system at a given point in time represents a correct state of the real world, a number of real-world constraints are expected to hold. In particular, your database schema must adhere to the following constraints:

- Constraints for Users
    1. No two users can share the same `User_ID`.
    2. All sellers and bidders must already exist as users.
- Constraints for Items
    3. No two items can share the same `Item_ID`.
    4. Every bid must correspond to an actual item.
    5. The items for a given category must all exist.
    6. An item cannot belong to a particular category more than once.
    7. The end time for an auction must always be after its start time.
    8. The `Current_Price` of an item must always match the `Amount` of the most recent bid for that item.
- Constraints for Bidding
    9. A user may not bid on an item he or she is also selling.
    10. No auction may have two bids at the exact same time.
    11. No auction may have a bid before its start time or after its end time.
    12. No user can make a bid of the same amount to the same item more than once.
    13. In every auction, the `Number_of_Bids` attribute corresponds to the actual number of bids for that particular item.
    14. Any new bid for a particular item must have a higher amount than any of the previous bids for that particular item.
- Constraints for Time
    15. All new bids must be placed at the time which matches the current time of your AuctionBase system.
    16. The current time of your AuctionBase system can only advance forward in time, not backward in time.

For the purposes of this Task, you can assume that only two types of modifications will be made to your database:

1. **The user may attempt to insert new bids.**
2. **The user may attempt to change the current time.**

You do not need to worry about any another types of modifications (e.g. insertion of new items, changing existing users, etc.) to the database.

As you can see, none of the constraints listed above are non-null constraints – you do not need to worry about these for this assignment, as you would probably need to enumerate a lot of them.

We have implemented these constraints for you. Here is how:

- **Constraint Design**: We have included a file `constraints.txt` – in this file, we specify, in plain English, how we implemented each of the 16 constraints in the database schema. Specifically, for each constraint, we state:
    1) How the constraint was implemented – did we choose to use a Key constraint (using `PRIMARY KEY` or `UNIQUE`), a Referential Integrity constraint, a `CHECK` constraint, or a Trigger? (Note that in SQLite, Referential Integrity constraints are often referred to as Foreign Key constraints.)
    2) Which file(s) contain the constraint implementation.
- **Implementation of Key, Referential Integrity, and CHECK constraints**: We have modified the `create.sql` file to include Key, Referential Integrity, and `CHECK` constraints.
- **Implementation of Trigger constraints**: We have also implemented some Trigger constraints – for each one, create two files:
    1) `triggerN_add.sql`

2) `triggerN_drop.sql`

where `N` $= 1, 2, \ldots, 8$.

In each `triggerN_add.sql`, we write the necessary SQL commands (using the `CREATE TRIGGER` syntax) to create the necessary trigger(s) that are needed to enforce that particular constraint. **Remember**: a Trigger constraint can potentially be violated by one or more types of database modifications, so we may need to write multiple triggers to properly enforce our constraints. The provided triggers handle the two types of modifications mentioned above – inserting a bid and changing the time. If a trigger discovers that a constraint is violated, it can either modify the database to somehow make the constraint hold, or it can raise an error. Errors within a SQLite trigger are raised by issuing the following `SELECT` statement:

```
SELECT raise(rollback, '<your error message>');
```

When this statement is executed, the modification command that activated the trigger is undone and the specified error message is printed.

Task C: Required functionality

You will know build the final AuctionBase system over the provided backend. The functionality of your final AuctionBase system is somewhat flexible (you can add additional functionality if you want). However, you must implement at least the following basic capabilities in order to receive full credit on the project:

- Ability to manually change the "current time."
- Ability for auction users to enter bids on open auctions.
- Automatic auction closing: an auction is "open" after its start time and "closed" when its end time is past or its buy price is reached.
- Ability to browse auctions of interest based on the following input parameters:
  - item ID
  - category
  - item description (This should be a substring search, i.e. not an exact match.)
  - min price
  - max price
  - open/closed status

  Note that these parameters are compositional, i.e. you should be able to browse by category **and** price, not category **or** price
- Ability to view all relevant information pertaining to a single auction. This should be displayed on an individual webpage, and it should display all of the information in your database pertaining to that particular item. This page should be linked to from your search results. In particular, this page should include:
  - all item attributes (title, description, etc.)
  - categories of the item
  - the auction's open/closed status
  - the auction's bids. You should also display all relevant information for each bid, including
    - the name of the bidder
    - the time of the bid
    - the price of the bid
  - if the auction is closed, it should display the winner of the auction (if a winner exists)

Furthermore, your AuctionBase system must support "realistic" bidding behavior. For example, it should not accept bids that are less than or equal to the current highest bid, bids on closed auctions, or bids from users that don't exist. Also, as specified above, a bid at the buy price should close the auction. **Some of these restrictions are already checked by the provided constraints and triggers; others may require additional triggers or code.**

If you do decide to add more triggers to your database, create additional `triggerN_add.sql` and `triggerN_drop.sql` files to implement these, and include them as part of your submission. You should also be sure to update your `createDatabase.sh` script to include these extra trigger files. (See the submission instructions at the end of this document for more details.)

Full credit also requires general error- and constraint-checking as specified in Task C below. For starters, all of the constraints provided should be checked in your "live" AuctionBase system.

Note that you can receive full credit on the project by implementing just the basic capabilities specified earlier, along with constraint-checking, error-checking, and a simple web interface. That is the standard against which projects will be graded. Many of you will realize that it is easy to add functionality and not difficult to enhance the user interface significantly. Feel free to be creative and unique! CS564 is not a user interface class and, again, you can receive full credit for a solid system with simple input boxes, menus, and simple HTML output tables. **However, under no circumstances should you be expecting the end-user to write SQL!**

Task D: Transactions, errors, and constraint-checking

Commands that modify the database need to be handled carefully, and you should group them into transactions whenever it makes sense for them to be executed as a unit. Using transactional behavior, each unit should either complete in its entirety or, due to failed constraints or other errors, should not modify the database at all. Constraint violations, and other errors due to bad input values or data entry, should be managed gracefully: It must be possible for users to continue interacting with the system after a constraint violation or error is detected, and the database should not be corrupt. You should inform users when errors occur, but your error message need not indicate the exact violation that caused the error.

If it helps, you may assume that AuctionBase has only one user operating on it at a time. Although transactions may be useful for database modifications and constraint-checking, you do not need to worry about transactions as a concurrency-control mechanism. That said, even without special effort your system may turn out to be fairly robust for multiple users.

Task E: Other Miscellaneous Requirements

- When you generate dynamic HTML pages from your program, please use relative paths, rather than absolute paths, for the links to the various URLs in your website. Relative paths enable us to grade your project in our own webspace.
- You don't have to implement user authentication. For example, it's okay to ask the user to enter his/her username when bidding, without asking for a password.
- Please make your website accessible by user interface. For example, a user should not have to "know" a URL and type it directly into the browser address bar. They should be able to click on links to navigate.
- Lastly, a suggestion: it's a good idea to debug your queries directly in SQLite before hooking them into your web interface. Use the SQLite command-line interface first, to ensure that your queries are working properly and are finishing in a reasonable amount of time. In the command-line interface, you can kill runaway queries using `Ctrl-C`. Once you are certain your queries are working properly, incorporate them into your web interface.

Submission instructions

Congratulations! You've completed the AuctionBase project! ☺

To submit your work, first create a submission directory with the following:

```
Your web.py/ directory (without your .db binary file!)
parser.py
runParser.sh
create.sql
load.txt
constraints_verify.sql
trigger{1..N}_add.sql
trigger{1..N}_drop.sql
createDatabase.sh
```

Finally, prior to submitting your project, you should perform the following to verify correctness:

1. Run your parser: `./runParser.sh`
2. Run your database creation script: `./createDatabase.sh`
3. Move your auctions.db database file into your web.py directory: `mv auctions.db web.py/`
4. Visit your auction website and confirm all of the features work correctly.
5. Delete your database file and any extraneous files (such as .dat) created from running your parser: `rm *{,/*}.{db,dat}`

In particular, note that we do **not** request that you include your `.db` database binary file as part of your submission. Instead, we will generate your database file using your `createDatabase.sh` script.

This means that you must include all necessary `*.sql` files that are required for `createDatabase.sh` to run properly! In particular, be sure to include any extra `triggerN_add.sql` and `triggerN_drop.sql` files that you may have added!

Also, if you made any other modifications **please also include those modified files as part of your submission**.

Once your submission directory is properly assembled, **with no extraneous files**, zip it and upload it to Canvas.