CS 145: Introduction to Databases

Stanford University, Fall 2015

AuctionBase Project: Database and the Web

Part 2: Data Integrity

Due Date: Tuesday, November 10th, 2:59pm

Overview

This part of the project will make use of SQLite constraints and triggers to monitor and maintain the integrity of your AuctionBase data. You will also add a "current time" feature to your AuctionBase database. *Because different database systems support different capabilities for constraints and triggers, this part of your project must be implemented in SQLite and validated on the Stanford Corn machines.*

Task A: Getting Started

Before you begin working on Part 2, first activate your personal CGI account on the Stanford Unix machines. Visit Stanford's Personal CGI Service page and follow the instructions to set up your CGI account. Once your account has been activated, you will have a new directory `~/cgi-bin/` in your personal AFS space, which you will need for Part 3 of the project.

Activation can take up to 24 hours, so we **strongly urge you to request your CGI account as soon as possible**, in order to avoid any potential delays when you begin working on Part 3.

Next, we highly encourage you to complete the optional Constraints and Triggers activity. The activity is not graded, but it provides all the details of constraints and triggers you will need to complete this part of the project. Once you have completed the activity, you can compare your solutions to the reference solutions.

Finally, take a look at the reference database schema for Project Part 1 posted on Piazza. You are free to use the schema you designed in Part 1, but the remaining parts may be more difficult (but certainly not impossible!) with an alternate design. Please note that switching to our schema will have no impact on your grade for Part 1.

Task B: 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 fully test your functionality, and to simulate the true operation of an online auction system in which auctions close as time passes, you should maintain a fictitious "current time" in your database. First, add a new one-attribute table to your 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. Later, in Part 3, when we ask you to simulate time advancing in AuctionBase, you'll do so by updating this table.

For starters, the table should be initialized to match the single point in time we've previously mentioned: 2001-12-20 00:00:01. To do this, modify your `create.sql` from Part 1 by adding the necessary SQL commands to create and initialize your "current time" table. Also, to make sure that you've initialized everything correctly, include a `SELECT` statement that reads the current time of your AuctionBase system. Your `create.sql` should now include the following:

```
DROP TABLE if exists CurrentTime;
CREATE TABLE CurrentTime(...);
INSERT into CurrentTime values (...);
SELECT ... from CurrentTime;
```

Task C: Adding Constraints and Triggers to Your Schema

Before getting started on 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.

Note: Depending on how you design your constraints, some of them may satisfy more than one of the requirements listed above. In this case, please still create a constraint for each requirement, even if they overlap in functionality.

Here is what you need to do:

- **Design your constraints**: Create a file called `constraints.txt` – in this file, you will specify, in plain English, how you implemented each of the 16 constraints in your database schema. Specifically, for each constraint, you need to state:
  1) How the constraint was implemented – did you 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 – this will be useful for us when we grade your Trigger constraints, if you have any.
- **Implement your Key, Referential Integrity, and CHECK constraints**: Once you've identified how each of the 16 constraints will be implemented, the next step is, of course, to implement them. Focus first on all of the constraints that will **not** be implemented using Triggers – modify your `create.sql` file once again to include your Key, Referential Integrity, and `CHECK` constraints.

  Once you've made your modifications, reload your database with the new constraints:

  ```
  /usr/class/cs145/bin/sqlite3 <db_name> < create.sql
  /usr/class/cs145/bin/sqlite3 <db_name> < load.txt
  ```

- **Verify your Referential Integrity constraints**: Unless you modified your `load.txt` file from Part 1 to begin with `PRAGMA foreign_keys = ON;`, the Foreign Key constraints specified in your `create.sql` file will not be enforced during bulk-loading, and this is for the best – performing a bulk-load while enforcing Referential Integrity can be very slow in SQLite. (All other constraints are checked during bulk-loading.)

  Therefore, you need to verify that the initial data actually satisfies your Referential Integrity constraints through some alternative means. Write a `SELECT` statement for each Referential Integrity constraint that returns an empty result *if and only if* the constraint holds. Create a file called `constraints_verify.sql` with all of your constraint-verifying `SELECT` statements:

  ```
  SELECT ... /* SELECT statement verifying Referential Integrity constraint #1 */
  SELECT ... /* SELECT statement verifying Referential Integrity constraint #2 */
  ...
  ```

  If any constraints do not hold, then either your constraints are incorrect, or your database is in an inconsistent state. **Stop now and fix the problem!**
- **Implement your Trigger constraints**: Now, let's implement your Trigger constraints – for each one, create two files:

1) `triggerN_add.sql`
2) `triggerN_drop.sql`

where $N = 1, 2, \ldots,$ etc., depending on how many Trigger constraints you determine are necessary.

In `triggerN_add.sql`, you'll 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 you may need to write multiple triggers to properly enforce your constraints. You should also make sure to 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. You can raise an error within a SQLite trigger 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.

Your `triggerN_add.sql` files should have the following format:

```
-- description: <constraint_description>
PRAGMA foreign_keys = ON;
drop trigger if exists <trigger_name>;
create trigger <trigger_name>
{before|after} {insert|update|delete} ON <table_name>
for each row
when <expression>
begin
   ...
end;
... /* add more triggers as needed */
```

And your `triggerN_drop.sql` files should have the following format:

```
PRAGMA foreign_keys = ON;
drop trigger <trigger_name>;
... /* drop additional triggers */
```

Lastly, don't forget to update your `constraints.txt` file as you implement your Trigger constraints – you should include both `triggerN_add.sql` and `triggerN_drop.sql` when listing which file(s) contain the implementation for a particular Trigger constraint.

Automating the process

Just like in Part 1, it helps to have a bash script that automates the process of creating your database, bulk-loading the data, and adding and verifying your constraints.

Create a file called `createDatabase.sh` which contains all of the necessary commands to create a database file for your AuctionBase system. This database file should have all of the data and your constraints loaded into it.

For example, your `createDatabase.sh` file might consist of:

```
/usr/class/cs145/bin/sqlite3 AuctionBase.db < create.sql
/usr/class/cs145/bin/sqlite3 AuctionBase.db < load.txt
/usr/class/cs145/bin/sqlite3 AuctionBase.db < constraints_verify.sql
/usr/class/cs145/bin/sqlite3 AuctionBase.db < trigger1_add.sql
... /* read in more trigger files as needed */
```

This will make it easier for you to test your database; it will also come in handy when you complete Part 3 of the project.

Submission instructions

To submit Part 2 of the project, first gather the following files in a single submission directory:

```
create.sql
load.txt
constraints.txt
constraints_verify.sql
trigger{1..N}_add.sql
trigger{1..N}_drop.sql
createDatabase.sh
{your_parser_name}.py
runParser.sh
```

Please make sure that each of these files is updated for any schema changes you've made for this part of the project.

Once your submission directory is properly assembled, **with no extraneous files**, execute the following script from from your submission directory:

```
/usr/class/cs145/bin/submit-project
```

Be sure to select "Part2" when the script prompts you for which assignment you're submitting!

As before, **do NOT include any data (.json, .dat, etc) or database files in your submission!** We reserve the right to deduct points from your project grade if you include them.

You may resubmit as many times as you like; however, only the latest submission and timestamp will be saved, and we will use your latest submission for grading your work and determining any late penalties that may apply. Submissions via email will not be accepted!