CS 434/534
Database Management Systems.
Cenek. Fall 2022.
Due Final Submission (suggested: December 9th 2022)
Final Project: Stage 2 **-- to be completed individually**

DEVAM PATEL

The final project's second stage is to go back to mysql database design and add all necessary database components to make the RDBMS fully functional and support the online auction application interface to the DB.

Consider the trade-offs of the DB theory vs. the data reality: so far, we focused on getting the data in, without enforcing many of the referential integrity constraints, database optimizations, transactions, etc. to support the online portal functionality. Now, let's add them to the database. This project is a full stack application development implemented on the LAMP server. Also, it is important that the data you have is only a data snapshot in time, and the database application has to support the future bidding on items.

The final project breakdown:
~~Stage 1: write a data parser and dump the data into respective flat files that represent database tables. This stage will also have rudementary DB design, as a simple check if your parser works. If the parser does not work, you won't be able to import the data into the DB.~~ DONE
Stage 2: complete the database design that includes all mechanisms to store, retrieve, age the sale items, and execute the transactions.
Stage 3: build a simple front end to interact with the auction database

As this assignment builds on the previous one, if you did not complete previous homework, you may request a copy of the raw, parsed data as .csv files for an automatic deduction of 30 points from the maximum possible points for this assignment (70/100 pts).

Part 1: Sanity check
Let's run some sanity checks on your data to make sure you will be able to implement the part 2 of this assignment and then build the front end of the full stack app.
This is not an exclusive list of checks to make sure your parse is built correctly.

1. Are the aggregate (calculated) fields in the items entity of integer type (for example ("number of bids"): Answer (circle one): [✓ Yes] / No

2. Do the aggregate bid counts you parsed from the json files match the number of bids records you have in your bids entity? Build a nested query that will group the bids by itemID and calculates the aggregate count and reports the difference between the calculated aggregate count and the value in the number of bids. Do the counts match: **yes, difference was 0**

Answer (circle one): [✓ Yes] / No

3. Are all the time fields imported as an appropriate numeric format, so you can run queries that compare two time fields/values?

Answer (circle one): [✓ Yes] / No

Build a query that will report the number of items for which the bidding closed (items auctioned off or sold) as of December 1, 2000. You most likely will have to convert date stamp from: "Nov-25-20 12:14:14" to ISO format: "2020-11-23 12:14:14" and re-import the data.
Answer: What is the number of records? **19531**

4. Similar to the above question 3, can you do the same for the currency comparison? In other words, can you check how many items are either sold or are currently for sale with the price of less than $95.76? What is the number of items for sale at this or lower price?
   Answer: What is the number of records? **19018 Items**

If you need to, and you have to go back to your parser and updated to fix the above definitions, this is the time to do it.

Deliverable:
a. Describe what fixes and alterations you made to the database and the parser.
Answer: Copy and paste the queries you used to alter the database with a one line description for each query.

I added a new table, tblItemCategory which holds the ItemID and its respective category ids. I deleted the CategoryID attribute from tblItems.

```
// new table which was added to the script
CREATE TABLE tblItemCategory (
ItemID int,
CategoryID int
);
```

The updates I made directly to script1.sh which was the original script that created the database and the tables. This UPDATED script1.sh is also included in the submission.

I changed my parser to add a new function getItemCategory() which creates a new csv file for itemID and corresponding categoryIDs.

b. Submit your updated parser.
c. Queries you defined to check the record correctness in your db: 1_2.sql, 1_3.sql, 1_4.sql,

Part 2: DB alterations
In addition to the mechanics of altering your DB, this part also tests and illustrates you abilities to design skills and problem solving skills. That's why I will not explicitly dictate what mechanisms the DB needs (I will try to give you helpful hints). I strongly encourage you to carefully inspect the last stage of front end design to extract the processes, details, actions that the DB has to handle.
Write a script called DBUpdateScript.sh that will alter the database Entity-Relationship design with the necessary database mechanism:

1. for each entity, create primary key(s) and foreign keys that will allow for cross-reference the entity with other entities in the database.
2. add the constrains associated with adding, removing and updating entities, relationships and attributes.
3. write trigger(s) that calculate the aggregate attributes in your entities (such as Number of Bids in the tblItems, etc). On the complexity of this aspect: you probably don't want to recalculate this attribute every time you have a user viewing the item, only when a valid bid is processes, the count should be updated.
4. write trigger(s) that will enable a valid authenticated user to bid on an item. The trigger(s) should include the following:

   1. check the time stamp to make sure that the bid processing is being executed on an item that is still up for sale
   2. check the item current price value to make sure that the bid processing is being executed on the bid amount that is greater than the current item sale price
   3. update the derived fields in the database (these will include: number of bids, current price, current highest bidder etc.
5. to speed up your queries, create and add indices on the query parameters you are asked to search for. The next assignment details the search functionality for description, category and seller. Please see the next assignment for details. The indexing will not solve your malformed queries, but rather optimize correctly working ones.
6. the above queries have to provide correct transactions processing mechanism to support the auction site concurrency – any and all actions on the data should be processed as atomic transaction so other user's requests will not leave the database in an inconsistent state – these actions include concurrent bidding etc. This feature is very hard to test, but I will look for its implementation.

MySQL supports stored procedures, which allows you to define and save the queries (also called procedures) inside of the MySQL. This way you will minimize your site's exposure to the sql injection attack. The sql queries will not be composed in the PhP, but simply called from the PhP by their stored name and executed with the parameters passed in. See the tutorial for details: https://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx I do not require this implementation, but it makes it a lot cleaner, functional, and safer.

Deliverable:
a. Describe what alterations and additions you made to the database. This section should cover the alternations suggested in Part 2 (note, part 2 is in no way exhaustive list of DB alterations).
Answer: Copy and paste the queries you used to alter the database with a one line description for each query.

Query: ALTER TABLE tblBids ADD CONSTRAINT bids_PK PRIMARY KEY (BidID);
Desc: Queries like above were used to add primary key to my tables.

Query: ALTER TABLE tblBids ADD CONSTRAINT bids_FK_users FOREIGN KEY (UserID) REFERENCES tblUsers(UserID);
Desc: Queries like above were used to add foreign keys to my tables.

Query: ALTER TABLE tblItems MODIFY COLUMN ItemName varchar(255) NOT NULL;
Desc: Queries like above were used to set constraint on columns to make sure they are required and not null when inserting new values

Query: ALTER TABLE tblItems ADD CONSTRAINT checkPrices CHECK (ItemID > 0 AND Currently >= 0 AND First_Bid >= 0 AND Number_of_Bids >= 0);
Desc: The above query was used to add a check constraint on ItemID, Currently, First_Bid, and NumBids to make sure they were all valid integers.

TRIGGER:

```
 START TRANSACTION; // TRIGGER IS WRAPPED IN A TRANSACTION

// create new trigger called newBigTrigger which executes BEFORE insert on tblBids
CREATE TRIGGER newBidTrigger BEFORE INSERT ON tblBids FOR EACH ROW
            BEGIN
                        SET @newTime = NEW.Time; // set newTime variable
  SET @endTime = (SELECT Ends from tblItems WHERE tblItems.ItemID = NEW.ItemID); // set endTime for the new item
  SET @currentAmount = (SELECT Currently FROM tblItems WHERE tblItems.ItemID = NEW.ItemID); // set current price
for new item
```

```
                    // if the new time and bid amoutn are valid, then update tblItems
                    IF @newTime < @endTime AND NEW.Amount > @currentAmount THEN
                              UPDATE tblItems
                              SET tblItems.Number_of_Bids = tblItems.Number_of_Bids + 1, tblItems.Currently =
NEW.Amount // update number of bids to reflect new bid
                              WHERE tblItems.ItemID = NEW.ItemID;
                    ELSE
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'ERROR: verify bid time or bid amount'; // otherwise return an error
message

                    END IF;
          END;

COMMIT; // end transaction


// Below are the indices I created to speed up data retrieval
CREATE INDEX idx_description ON tblItems (Description);
CREATE INDEX idx_category ON tblCategory (Name);
CREATE INDEX idx_seller ON tblItems (Seller);
```

What to turn in:
On moodle upload an archive with a .pdf copy of this document included with the final project submission. Fill out the "Deliverable" sections of the Part 1 and Part 2 above.