SQL – Relational 2 Assignment

Before you begin please do the following:

1. Watch the video lecture posted. Many of the commands I will show you on the video.
2. You can also look at the SQLCheatsheet under "Resources" for many of the common SQL commands.
3. The below exercises corresponds to the lecture titled Relational Model.

Submission:

You must create a SQL file with all your answers: The SQL file should have your name in it as a comment (using #) on the first line. The following line should have your "use" command preceded by a "#" comment. The next line will contain the date and number **that results** from: **`SELECT @d:=NOW(),SHA(@d);`** separated by a space or TAB. This line should also be preceded by a "#" comment, followed by a blank line.

Then, for every answer, first write a "#" followed by the question number and in the following lines write your answer. Then, skip a blank line and repeat the process for the next question.

If a question has a sub-item with further sub-items, only the top two levels comprise the question number. For example, if question 4 has two parts (4.1 and 4.2) and 4.1 has 3 bullets (a,b and c), paste your response under 4.1. Do not divide this into 4.1.a, 4.1.b and 4.1.c.

If you want to add additional comments, use the -- notation. You should be able to load your entire SQL file into your IDE, uncomment the second line and run your code without **syntax** errors, at least, before submitting. Remember to comment it back prior to submission.

You will need to submit the SQL file. **Other formats will not be graded**. The SQL file ends in **.sql** you can choose whatever name you like WITHOUT spaces. A good idea is your last and first name and homework number. For example: `DoeJohnHW2.sql`

Once your homework is formatted, please use the format checker to ensure that the very basics of the format are ok. The format checker is not completely precise, but can help you get rid of all major formatting mistakes.

For example, a file from John Doe with 2 questions, where the first question has 2 items, may look like this:

```
# John Doe
# use jdoe_db;
# 2017-01-05 17:38:27 41860934238f8b3a1fdd51dee11e2d4219c92068

#1.1
Nothing but SQL code goes
here as the answer to
The question. SELECTs, CREATEs, whatever.

#1.2
More SQL code answering
question number 1.2 (or I.2) and all its sub points (a,b,c,d, etc.)
The answer can take up as many lines as needed in SQL only
-- non sql line to explain a result (if asked)
-- should be commented by preceding it with -- and a space.

#2.1
Yet more SQL commands
That answer
The first part of the second question
```

Exercise:

The following assignment builds up on the previous one. We will use all the same tables and

1 Primary and Foreign Keys

1. At this point your application is really useful, but you can see the number of accounts growing exponentially if you were to track them in more detail. For example, if you want to know details like money spent on individual restaurants or individual items related to your car expenses (Maintenance, Gas, Tires, etc.) you would need to create a new account for every restaurant, car shop and gas station. While this gives you a lot of detail, it makes it very hard to track money spent globally on expenses categories, such as the total spent on clothes, car expenses or food expenses. Therefore, you want to categorize accounts so that accounts are specific within a category and, therefore, easy to track. For example, in the category “Food” I can have accounts such as Descartes Coffee, Starbucks and Olive Garden. In the category “Auto” I may have accounts such as “Pete’s

Body shop" or "Gas" or "Jiffy Lube". In the category "Clothes" I can have accounts such as "GAP," "Old Navy" and "Nordstrom". You get the idea. For this you are going to create a new table called "Category" with the following columns:

   a. *category_id*, an integer that auto increments
   b. *description, a varchar of length 64*
   c. *date_created a DATETIME*
   d. *budget_warn a CHAR(1) that will indicate Y for categories over budget and N for categories under budget. The default should be "N"*

   This table has a primary key which is the category_id.

   In addition, we will modify the Acct table and will add a column called category_id with a default of 0.

2. When inserting to an auto increment field, you can omit the attribute in the insert command, but you still have to provide the other field's values. Please insert a category with description "Food." And date_created of NOW().
3. Modify Acct. Change account 4 from 'Food' to 'Descartes Coffee Shop'. Create two new accounts of type 1 in Acct. The first, with id 8 called "McDonalds" and the second one with id 9 called "Starbucks". Accounts 4,7 and 8 should have a category_id of 1.
4. You realized that you did not have a category name for "0", which is the default. You see this as a potential violation of data integrity, so you have to create a foreign key to tie Acct to Category. The name of your foreign key shoud be **fk_ci**. Paste your ALTER TABLE command.
5. Was there a problem? If yes, paste the message and what you think is the reason in comments preceded by - -
6. Create a new category called "General". Then, update the default category_id in Acct to match this new "General" category. Lastly, update all the records in Acct where the category_id is zero and change them to the category_id of "General."
7. Try to alter the table and create the foreign key again. Paste your SQL statement. If there was a problem, paste it in a comment.
8. Insert a row in Acct with account_id 10, the description is "GAP" and the category_id is 3. Paste your statement and in comments, say if any errors happened and why.
9. Now, It makes sense that budgets are assigned to categories of items and not individual accounts. For example, you want a budget for expenses on Food. Not specifically at Descartes Coffee or a restaurant you went to once. So:
   a. Remove the budget_id attribute from the Acct table and add it to the category table
   b. Create a foreign key called **fk_cb** in category that refers to budget on budget_id.
   c. For the sake of completeness, create two foreign keys on Journal. **fk_from** on from_acct_id that references Acct's acct_id and **fk_to** on to_acct_id that

references Acct's acct_id. That way you won't be able to create transactions on non-existing accounts.

10. Now, you need to process all the necessary SQL statements for the following situation: There was a transaction from "Cash" to "GAP" for $95. "GAP" is an account associated with a category "Clothing" and the budget_id 1 ($300). There is another transaction from Cash to "Old Navy" for $180. Old Navy is a new account also associated with the category "Clothing."
11. Paste a command that selects the account names and amounts of all journal entries for the "Clothing" category. This way you can verify your previous commands worked correctly. *Hint: This is a select with multiple JOIN clauses (JOIN table ON condition JOIN table2 ON condition, etc.). Hint 2: You may have to join the same table more than once and assign it an alias. –that is*
12. You no longer want to track College expenses (acct_id 6), please delete the college account from Acct. Paste any error messages you get in comments --.
13. Delete the College Journal entry and the account. Paste the commands in the exact order they should be executed.
14. Because categories and budgets are so intertwined, you decide to create a foreign key that will remove categories if their corresponding budget is deleted. Remove the current foreign key and create a new one with the same name: **fk_cb** that "cascade deletes" records from the category table if a budget associated with a category is deleted (*Hint: Use ALTER TABLE for all your commands*). Paste the SQL commands in the exact order they need to be executed.
15. Associate a new category "College Expenses" with Budget id 3. Then Delete the budget 3.
16. Paste two select statements to verify that the category "College Expenses" is gone, as well as the budget #3.

II Triggers

1. You can imagine that inserting a transaction and checking whether you are over budget every time can be very time consuming. Create a trigger called **tg_journal** on the table Journal so that when you insert a transaction, a computation is done where you check whether the total transactions **TO** accounts of the same category are above or below budget. *Hint: only use the to_acct_id field here. We care about over spending* ON things. *Hint 2: Think of this as a process. First, get the category of the product, then the total spent on that category, then the budgeted amount for the category. Then, compare both and using an IF THEN ELSE END IF, decide whether you will update the category with a Y or an N.*

2. Now, insert a transaction from cash to "GAP" for $100.

3. Paste a select statement to verify that the trigger worked.

III Indices

1. At some point, you will be looking into accounts based on their names. Because you are looking in a potentially large table that can get slow over the years, Create an index called **idx_acct** on description in Acct so that the join does not take so long[1]. Paste the SQL command.

[1] In this example you will not be able to tell the difference, but in large datasets this difference can be quite noticeable.