Fall 2017 Comp 533
Assignment 4
Due: November 14, 2017 at 11:45 PM

The goal of this assignment is to write triggers and stored procedures in our database.

Please make sure that every time you run the set of tests associated with a task, you use a "fresh" copy of the database. If you don't, you are likely to get the wrong answers. You can drop and recreate the tables, or just truncate and reload them. You can even dump and reload your database. Just be careful to not lose any of the code you have written.

What to turn in

You must turn in a .sql file on Canvas. It must include all SQL code, including the "run this code to check your answer" statements.

Grading

Since the tasks increase in complexity, they are worth different amounts of points.

Within each task the percentage of points will be allotted based on the following guidelines:

- 0: Task not attempted, Task does not give any results, or it does not compile
- 25%: Task compiles, runs and is most of the way towards a correct answer, but there are no comments
- 75%: The task and answer it produces are almost correct, but there is a slight or subtle bug in the task **or there are limited or no**

comments.

- 100%: The task is correct, gives the right answer and is clearly documented

The survey at the end is worth 5 points.

What's In and Out of Scope

This is intended to be a programming assignment. While it may be possible to implement some of these tasks in pure SQL, that is not the intent. You must use transact-SQL, other than for creating and altering tables and retrieving results.

Comment your code! Explain what you are doing. You must have comments to get full credit.

Academic Honesty

The following level of collaboration is allowed on this assignment: You may discuss the assignment with your classmates at a high level. What is not allowed is direct examination of anyone else's code (on a computer, email, whiteboard, etc.) or allowing anyone else to see your code.

You may use the search engine of your choice to lookup the syntax for transact-SQL and SQL commands, but may NOT use it to find solutions to the tasks, either as guidance or for actual transact-SQL.

You MAY post and discuss results with your classmates.

1 Create Tables

We're going to start with a clean copy of the database. Execute the following code to remove existing tables and recreate them. Be sure to backup your database, rename the tables or export content if you want save anything you have created. The data is different from the last assignment. So be sure to reload the data!

```
DROP TABLE IF EXISTS itemSold;
DROP TABLE IF EXISTS productSold;
DROP TABLE IF EXISTS menuItem;
DROP TABLE IF EXISTS eventAssignment;
DROP TABLE IF EXISTS ticket;
DROP TABLE IF EXISTS employee;
DROP TABLE IF EXISTS event;
DROP TABLE IF EXISTS menu;
DROP TABLE IF EXISTS location;
DROP TABLE IF EXISTS recipeItem;
DROP TABLE IF EXISTS product;
DROP TABLE IF EXISTS component;
DROP TABLE IF EXISTS unit;
DROP TABLE IF EXISTS componentCategory;
```

Recreate the tables from the file Assignment4tables.sql.

If you accidentally load the data more than once, or run into some other problem, you can rerun these steps after dropping the tables and recreating and loading them as described above. If you have to repeat the process, either DROP or TRUNCATE the table and restart. If you truncate the table, use the follow command to reset the auto number sequence:

ALTER TABLE <tablename> AUTO_INCREMENT = 1;

2 Triggers & Functions

Answer all of the questions below by writing and executing triggers, functions, stored procedures, and any supporting SQL statements and queries.

2.1 Ticket Price Trigger, 10 points

In an earlier assignment, we wrote an update statement to populate the totalPrice field in the ticket table.

That approach was useful for a one time update. However, we want the

totalPrice field to be updated EVERY time we sell a product.

Write a trigger named 'updateTicket' on the productSold table that updates the totalPrice and numProducts fields in the ticket table for the corresponding ticket when a productSold record is added.

Check the ticket information before adding another product:
SELECT * FROM ticket
WHERE ticketId = 187;

Add another product to the ticket:
INSERT INTO productSold(productCode, ticketId) VALUES ('bs', 187);

Check the ticket information after adding the brownie sundae:
SELECT * FROM ticket WHERE ticketId = 187;

2.2 Event Booking Trigger, 20 points

Owl ice cream doesn't want to accidentally over book events.

Add a 'valid' field to the event table, which has a default value of 0. If this field is set to 1, the event is valid. If it is 0, the event is not valid and the owner has to decide what to do.

Set the value of this field to 1 for all existing records in the table. Provide the SQL statement to do this!

Next, add an insert trigger(s) to the event table that will insert the new record, but set the valid value to 0 if the new event overlaps with any existing event.

We need to make it easy to see what existing events conflict with the new event(s). Let's put the conflicts in a table, since we can't print out statements in a trigger.

Create a table named 'eventIssue'. This table should contain 9 fields - the id, name, start and end datetimes of the newly inserted event, the eventId and eventName, start and end datetimes of all existing, valid events that conflict with the new event and a status field. The status field can take on 2 values: 'overlap' and 'travel'.

Use the following attribute names:

newEventId

newEventName

newStart

newEnd

existEventId

existEventName

existStart

existEnd

status

When an overlap conflict is found, insert the appropriate record into eventIssue and set the status field value to 'overlap'.

Also, check to see if there is a limited (less than or equal to 30 minutes) travel window between the new event and any existing events. If there is, create rows in the eventIssue table for these issues as well. For these events, set the status value in the eventIssue table to 'travel'. Set the valid value in the event table for the new event to 1, if there isn't an overlap issue.

Test your trigger by executing the following code:

```
INSERT INTO event(eventName, eventStart, eventEnd, locationId, menuId)
VALUES
('Mid afternoon tea', '2017-03-09 14:15:00', '2017-03-09 14:45:00', 24, 1),
('Dessert Bar', '2017-01-19 13:30:00', '2017-01-19 16:00:00', 25, 4),
('Ice cream for lunch', '2017-05-06 11:00:00', '2017-05-06 16:00:00', 17, 3);
```

Check your results:

```
SELECT *
FROM event
```

WHERE eventName in ('Dessert Bar', 'Ice cream for lunch', 'Mid afternoon tea')
ORDER BY eventName;

SELECT *
FROM eventIssue
ORDER BY newEventName;

2.3 Restocking Tasks, Function 5 points, Trigger 30 points

There are tasks that need to happen during events to maintain inventory levels. We are going to write records to a table that could be polled frequently to find out if we need to restock any supplies.

Create a table named 'eventTask' and a table named 'invLevel'.

EventTask should contain an eventId, a task type, a componentCategoryId, a quantity and the datetime the task was performed.

InvLevel should contain a componentCategoryId and a restock quantity.

Populate InvLevel with the following data:

CompCatId	RestockQty
5	200
6	50
13	50
14	50
16	300

FYI, these component categories are: paper products; plastic products; spoons; cups; and straws.

Create a function 'getEventId' that takes a productSoldId and returns the eventId where that product was sold.

Test with the following code:
SELECT getEventId(3);

SELECT getEventId(6471);

Next, create a trigger that adds a 'restock' task type record to eventTask when the threshold number of components have been sold, as specified in the invLevel table.

Test with the following code:
INSERT INTO itemSold(productSoldId, compId, qty, unitId) VALUES
(6471, 48, 1, 3),
(6683, 46, 1, 3),
(6726, 46, 1, 3),
(7529, 48, 1, 3);

SELECT * FROM eventTask;

2.4 Event rescheduling Stored Procedure, 30 points

Answer this problem below by writing and executing a STORED PROCEDURE and any supporting SQL statements and queries.

Rice wants students to be healthier and decides to only let dessert food trucks be on campus for a maximum of N days in row.

We want to know which events would have to be rescheduled.

Write a stored procedure named 'eventsToCancel' that populates a table named 'badEvent'(eventId, eventName, eventStart) with a list of events that have to be rescheduled.

Test your stored procedure by executing the following code, running the results for N = 3 and N = 4.
DELETE FROM badEvent;
CALL eventsToCancel(3);

SELECT * FROM badEvent ORDER BY eventName;

CALL eventsToCancel(4);

SELECT * FROM badEvent ORDER BY eventName;

3 Survey (5 points)

It took me approximately N hours to complete this assignment, where N is: