# MSiA-413 Introduction to Databases and Information Retrieval

Homework 8: Triggers, Integrity Constraints, and Window Functions

Name 1: Sam Swain

NetID 1: sms5736

Name 2: Yaasir Ahmed

NetID 2: yap80676

# Instructions

You should submit this homework assignment via Canvas. Acceptable formats are word files, text files, and pdf files. Paper submissions are not allowed and they will receive an automatic zero.

As explained during lecture and in the syllabus, assignments are done in groups. The groups have been created and assigned. Each group needs to submit only one assignment (i.e., there is no need for both partners to submit individually the same homework assignment).

Each group can submit solutions multiple times (for example, you may discover an error in your earlier submission and choose to submit a new solution set). We will grade only the last submission and ignore earlier ones.

Make sure you submit your solutions before the deadline. The policies governing academic integrity, tardiness and penalties are detailed in the syllabus.

**SchoolScheduling.sqlite Database (50 points)**

The problems below will update some of the databases’ data, so first make a copy of the databases and use only that copy for the sample problems below. The questions must be done in order (each depends on the previous ones).

1. **(1 points)** First, in preparation for this practice, write and execute a query that inserts a new class status with ID 4 and description “Failed”.**﻿**

**insert into Student\_Class\_Status (ClassStatus, ClassStatusDescription)**

**values (4, "Failed")**

1. **(1 points)** Student 1001 is taking class 4180. When the quarter finishes, the class’ status in the student’s schedule should be updated to “Completed”, i.e., set classStatus=2. Write and execute a query that performs the update.

**update Student\_Schedules**

**set ClassStatus = 2**

**where StudentID == 1001 and ClassID == 4180**

1. **(1 points)** The problem now is that if the administrator forgets to set the grade, the student will have a class marked as completed with grade 0. You want to enforce a rule that a class cannot be marked completed unless the student receives a passing grade, i.e., a grade of at least 60.0.

First, write and execute a query that reverses the previous change you made in question 2, so you can try again later.

**update Student\_Schedules**

**set ClassStatus = 1**

**where StudentID == 1001 and ClassID == 4180**

1. **(22 points)** Now, write the query that implements the data integrity constraint described in question 3.

**create trigger to\_completed\_class**

**before update**

**on**

**Student\_Schedules**

**for each row**

**begin**

**select case**

**when old.Grade < 60.0**

**and new.ClassStatus == 2**

**then raise(fail, "Error: Grade must be greater than or equal to 60.0")**

**end;**

**end;**

1. **(25 points)** Write a query that enforces the following data integrity constraint: when a grade changes from a non-passing grade to a passing grade, automatically set the status of the class to completed. If the grade changes to a non-passing grade, set the class status to failed.

**create trigger update\_class\_status**

**after update on Student\_Schedules**

**for each row**

**begin**

**update Student\_Schedules**

**set ClassStatus =**

**case**

**when Grade >= 60.0 then 2**

**when Grade < 60.0 then 1**

**end;**

**end;**

**Homework 5 Question 6 Solution Database (25 points)**

1. **(25 points)** Sometimes it is hard (or impossible) to implement integrity constraints using triggers or the CREATE TABLE command. In such cases, one solution is to create a view that you will examine whenever you want to verify that your database conforms to the underlying assumptions of your data. One such example are the constraints (e), (g), (i), (k), and (n) in Homework 5 Question 6.

For this assignment, use the Homework 5 Question 6 solution database and create a view named check\_db that checks if the database violates any of the constraints (e), (g), (i), (k), and (n). The view should return a table that lists all the violated constraints, or an empty row if there are no violations. For example, if constraints e, g, i, k, and n are all violated, the view will be the table below (do not worry about the order of rows):

Table

Description automatically generated

Note that it is OK if some of the rows of your result are empty rows. Similarly, if no constraints are violated, the view could simply return a table with an empty row:



To check that your view works properly, you can execute the following deletions on the Homework 5 Question 6 solution database and check the output of your query after each deletion set, as the comments and the SQL queries below show. The queries should be executed in the exact order below to achieve each of the stated results.

Suggestion: you can use BEGIN TRANSACTION and ROLLBACK when you are experimenting in this question. This way, when things don’t work, you can simply rollback the changes. Note that if you rollback once you will need to start a new transaction again to be able to rollback your changes a second time (ROLLBACK will undo your changes AND terminate the transaction). Even better, use SAVEPOINT X and ROLLBACK TO X. This way you can issue ROLLBACK TO X as many times as you want without taking a new savepoint (ROLLBACK TO X will undo the changes but the transaction remains active, so there is no need to remember to start a new one each time).

-- check that your view works by examining it on the HW5 Q6 solution database before any deletions

-- your view should be just an empty row (i.e., there are no violations)

SELECT \* FROM check\_db;

-- performing the following deletions would violate the following constraint

-- k. Each invoice has at least one invoice item

-- your view should contain a row for k

DELETE FROM invoice\_items WHERE invoiceId IN (2, 3);

SELECT \* FROM check\_db;

-- performing the following additional deletions would violate the following additional constraint

-- e. Each album has at least one track

-- your view should contain rows for e, k

DELETE FROM tracks WHERE albumId=2;

SELECT \* FROM check\_db;

-- performing the following additional deletions would violate the following additional constraint

-- g. Each genre is represented by at least one track

-- your view should contain rows e, g, k

DELETE FROM tracks WHERE trackId=3451;

SELECT \* FROM check\_db;

-- performing the following additional deletions would violate the following additional constraint

-- i. Each media type is used by at least one track

-- your view should contain rows for e, g, i, k

DELETE FROM invoice\_items WHERE trackId IN (SELECT trackId FROM tracks WHERE mediaTypeId=4);

DELETE FROM tracks WHERE mediaTypeId=4;

SELECT \* FROM check\_db;

-- performing the following additional deletions would violate the following additional constraint

-- n. Each customer has been issued at least one invoice

-- your view should contain rows for e, g, i, k, n

DELETE FROM invoice\_items WHERE invoiceId IN (SELECT invoiceId FROM invoices WHERE customerId=20);

DELETE FROM invoices WHERE customerId=20;

SELECT \* FROM check\_db;

**CREATE VIEW check\_db AS**

**SELECT \* from (**

**select Case when count(DISTINCT a.albumid) != count(DISTINCT t.albumid) then "e" else "" end ERRORS\_FOUND**

**from albums a, tracks t**

**union all**

**select Case when count(DISTINCT t.genreid) != count(DISTINCT g.genreId) then "g" else "" end ERRORS\_FOUND**

**from tracks t, genres g**

**UNION ALL**

**select Case when count(DISTINCT t.mediatypeid) != count(DISTINCT m.mediatypeid) then "i" else "" end ERRORS\_FOUND**

**from tracks t, media\_types m**

**UNION ALL**

**select Case when count(DISTINCT ii.invoiceid) != count(DISTINCT i.invoiceid) then "k" else "" end ERRORS\_FOUND**

**from invoice\_items ii, invoices I**

**union ALL**

**select Case when count(DISTINCT c.customerid) != count(DISTINCT i.customerid) then "n" else "" end ERRORS\_FOUND**

**from customers c, invoices I**

**)**

**SalesOrders.sqlite Database (25 points)**

1. **(25 points)** Monthly revenue growth is defined as the percent of revenue change of a month relative to the previous month, i.e., (Mi - Mi-1) / Mi-1. Write a query that will return the revenue growth of the sales in the SalesOrders database. This should be a single query (CTE and windowing allowed).

**with revenue\_growth(Month, RevenueAVG) as (**

**select**

**date(orderdate, 'start of month', '+1 month', '-1 day'),**

**sum(ordertotal)**

**from Orders**

**group by date(orderdate, 'start of month'))**

**select**

**Month,**

**round((RevenueAVG - lag(RevenueAVG) OVER(ORDER BY Month))/lag(RevenueAVG) OVER(ORDER BY Month)\*100,3) as Growth**

**From revenue\_growth;**

**Output: ON NEXT PAGE**

