EECS 484 Homework #2

(96 points)

Due: Friday, Oct 4th, 2024 at 11:45 pm (ET)

There are two parts in the homework assignment. Please follow the guidelines and upload all your query files (12 in total) to the Autograder. There is no limit to the number of submissions you can make every day, but only **the first two submissions per day will receive score feedback**. You are responsible for testing queries in the SQLplus environment on CAEN before making any submissions. Note that although the Autograder states that there are 105 possible points to earn, **a score of 96 is the maximum attainable score and is considered a full score**. This is due to the fact that you can earn partial credits on certain queries, a configuration which cannot be displayed properly on the Autograder. There are no hidden test cases.

You must complete this assignment individually. You may not share answers with other students actively enrolled in the course, nor may you consult with students who took the course in previous semesters. You are, however, allowed to discuss general approaches and course concepts with other students, and you are also permitted (and encouraged!) to post questions on Piazza. As always, you are also encouraged to come to Office Hours with questions.

The University of Michigan College of Engineering Honor Code strictly applies to this assignment, and we will thoroughly check to ensure that all submissions adhere to the Honor Code guidelines. Students whose submissions are found to violate the Honor Code will be directly reported to the Honor Council.

Late submissions for this assignment will not be accepted (project late days do not apply to homework). The Autograder enforces a strict cut-off at the specified deadline, so be sure that your submission is made prior to that time. Your best submission will be used for grading.

Part 1: University SQL (40 points)

Consider a database consisting of the following five tables. Attributes that are underlined have been designated the primary key of their respective tables, and fields with identical names in different tables can be safely assumed to be foreign keys. The Project Name and Course Name fields are specified as UNIQUE, and all fields are required *except* for Major, which may be NULL (undeclared).

Students(SID, Name, Major)

Projects(PID, P_Name)

Courses(CID, C_Name)

Members(PID, SID)

Enrollments(CID, SID)

Write each of the following queries in a separate file named **UniversityQuery[N].sql**, where [N] is the number of the query as enumerated. Each query is worth 10 points. Fields must be selected in the order they are given at the beginning of each problem statement and results must be sorted as specified at the end of each problem statement. We have not provided you with any sample data for this part of the assignment, so it is suggested that you devise a way to test your scripts to ensure that they are correct. There are no restrictions on the methods you use to complete the query: nested queries, views, and set operations are all legal. **In all instances, do not have duplicate rows in your results.**

1. Write a query that returns CID of CS-heavy courses, which is defined as the following: strictly fewer than 10 non-CS majors are enrolled (including courses in which 0 non-CS majors are enrolled). A student whose major is CS will have the VARCHAR2 value 'CS' for their Major field, while a non-CS student will have something else. Remember that the Major field *can* be NULL. The results should be sorted in **descending** order by CID.

2. Write a query that finds the SIDs and Names of all students with at least one project partner who is enrolled in (EECS 482 or EECS 483) and (EECS 484 or EECS 485) and (EECS 280). Students who are Members of the same Project are considered project partners (a student is not a project partner of themselves). Note that each course's name is stored in the C_Name field with no space between the department abbreviation and the course number (e.g. EECS 484 is stored as 'EECS484'.) The results should be sorted in **descending** order by the students' names.

   Hint: you should be able to implement this query without using views or set operators (Union, Minus, Intersect).

3. Write a query that finds the list of the SIDs of all students who are enrolled in (EECS442 and EECS445 and EECS492) or (EECS482 and EECS486) or (EECS281). Do not return duplicates. The results should be sorted in **ascending** order by SID.

4. Create a view called **StudentPairs** with two columns, SID1 and SID2. The contents of this view should be all pairs of SIDs of two students who are enrolled in at least one common class but are not already partners on any project. You should report each student pair exactly once. The lower ID should be SID1, and the higher ID should be SID2. The contents of the view do not need to be sorted, and you do not need to drop the view. You **must NOT** create views other than the **StudentPairs** view.

Part 2: Booktown SQL (56 points)

A new bookstore called Booktown has opened in your suburb, and they've hired you to find some information about their current inventory. They've provided you with a database of their store's contents in **BuildBooktown.sql**, which defines the schema of the data tables as well as the data they contain. You should familiarize yourself with the schemas contained in this file, as they may contain information that is helpful in completing your queries. You can use the script file **DropBooktown.sql** to drop the database once you've finished using it.

Booktown wants to emphasize a few things about the schema of its database that you may accidentally overlook. Pay close attention to these points:

- There is no requirement that the pair ("First Name", "Last Name") for an author is unique; instances of the same pair with different "Author ID"s represent different authors
- Despite consisting only of numerals, the "ISBN" field of an edition is a string
- The "Publication Date" field is a string that has the format YYYY-MM-DD, with the month and day being padded with a leading 0 where necessary. These values can be compared using standard comparators (<, >, =)

Write each of the following queries in a separate file titled **BooktownQuery[N].sql**, where [N] is the number of the query as enumerated. Each query is worth 7 points. As with part 1, fields must be selected in the order they are given at the beginning of each problem statement and results must be sorted as specified at the end of each problem statement. There are no restrictions on the methods you use to complete the query: nested queries, views, and set operations are all legal. Your queries must be correct *for any data set* and not just the one that we have provided; there is no guarantee that we will run your scripts against the exact data you have access to.

Be sure not to repeat results in the output of any of your queries. Be careful, though: just because the values of the fields you returned are the same in more than one row does not necessarily mean that they are the same result. This might happen if you are selecting only fields that are not guaranteed to be unique, as repeated tuples may then represent disparate entities.

1. Write a query that finds the IDs of all authors who have written exactly 1 book. The results should be sorted in **ascending** order.

2. Write a query that returns the "Subject" attribute of all subjects for which no book has been written by any author. The results should be sorted in **ascending** order.

3. Write a query that finds the ISBNs of all editions of books written by Agatha Christie. The results should be sorted in **descending** order.

4. Write a query that finds the first and last names of all authors who have written at least one children's/young adult book (subject: "Children/YA") and at least one book of fiction (subject: "Fiction"). The results should be sorted first in **ascending** order by first name and then in **ascending** order by last name. Note: if there are multiple authors with the same first and last name, their names should appear multiple times in the result.

5. Write a query that finds the IDs, first names, and last names of all authors who have written at least one book in every subject for which J. K. Rowling has written at least one book, including J. K. Rowling. You may assume there is only one author with a first name 'J. K.' and last name 'Rowling'. The results should be sorted in ascending order by the author's last name, with ties being broken in favor of the larger ID.

6. Write a query that finds titles, publication dates, author IDs, author first names, and author last names of **ALL** editions of books written by authors who have written at least one book with at least one edition published between the dates of 2003-01-01 and 2008-12-31. Both dates should be included in the range. The results should be sorted in ascending order by the author's ID, then in ascending order by the book title, then in descending order by the date of publication.

7. Write a query that finds the titles of books and the sum of pages across all editions of each respective book. The column containing the cumulative total should be named Total_Pages. Only books that have editions listed need to be included in the results, which should be sorted in descending order by the cumulative total number of pages across all editions.

8. Write a query that finds the IDs and names of publishing companies that have published at least one edition of a book written by any author who has written exactly 3 books. The results should be sorted in descending order by the publisher's ID.

Please make sure that all of your queries execute in SQL*PLUS without syntax errors or undesired side effects. If you ever create an intermediate view, be sure that you drop it at the end of the file in which it was created. Under no circumstances should you alias the names of the columns *except* in Query 7 as specified. Do not include any column formatting, SQL prompts, or other content in your scripts that might change the format of your output. Do not tamper with the autocommit feature either.