Project 1

Problem 1: Social Network Analysis

We provided a simple database for a social network, social.db . Here's the schema:

- Student(**ID** int, **name** text, **grade** int)
- Friend(**ID1** int, **ID2** int)
- Likes(**ID1** int, **ID2** int)

In the **Student** table, **ID** is a unique identifier for a particular student. **Name** and **grade** correspond to the student's first name and grade. In the **Friend** table, each (**ID1** , **ID2**) pair indicates that the student with **ID1** is friends with the student with **ID2** (and vice versa). The friendship is mutual, and if (**ID1** , **ID2**) is in the table, it is guaranteed that (**ID2** , **ID1**) exists in the table. In the **Likes** table, each (**ID1** , **ID2**) pair indicates that the student with **ID1** likes the student with **ID2** . The (**ID1** , **ID2**) pair in the table does not guarantee that the (**ID2** , **ID1**) pair also exists in the table.

1. Write a SQL statement that returns the names of students who are in grade 9. Results should be ordered by name (AZ).
2. Write a SQL statement that returns the number of students in each grade. The query should return 2 columns: grade, count. Results should be ordered by grade (ascending).
3. Write a SQL statement that returns the name and grade of all students who have more than 2 friends. Results should be ordered by name (AZ), then grade (ascending).
4. Write a SQL statement that returns the name and grade of all students who are liked by at least one student who is in any grade above. Results should be ordered by name (AZ), then grade (ascending).
5. Write a SQL statement that returns the name and grade of all students who only like their friends. You should also return students who don't like anyone as well. Results should be ordered by name (AZ), then grade (ascending).
6. Write a SQL statement to find pairs (A,B) such that student A likes student B, but A is not friends with B. The query should return 4 columns: ID1, name1, ID2, name2. Results should be ordered by ID1 (ascending), then ID2 (ascending).
7. For each pair (A,B) in the previous problem, find all the students C who can introduce them (C is friends with both A and B). The query should return 6 columns: ID1, name1, ID2, name2, ID3, name3. Results should be ordered by ID1 (ascending), ID2 (ascending), then ID3 (ascending).

Problem 2: Course Evaluation Data Analysis

A group of anonymous hackers have leaked the course evaluation data of the CS department at Grey University (our long time rival!). Turns out, they were using the Star schema for their database:

- Fact_Course_Evaluation (**professor_id** int, **term_id** int, **student_id** int, **type_id** int, **class_score** int, **efficiency_score** int, **content_score** int)
- Dim_Term (**id** int, **year** int, **term** text (possible values are 'spring' and 'fall'))
- Dim_Student (**id** int, **name** text, **college_year_name** text, (possible values are 'freshman', 'sophomore', 'junior', 'senior') **undergraduate** bit)
- Dim_Professor (**id** int, **name** text, **gender** text, (possible values are 'male' and 'female') **tenured** bit)
- Dim_Type (**id** int, **type** text, (possible values are 'course', 'seminar' and 'independent_study') **area** text (possible values are 'ai', 'systems', 'theory'))

The database is provided as university.db .
1. Write a SQL statement to find out if tenured professors get better class scores than untenured professors. The query should return the average class scores for tenured and untenured professors.
2. Write a SQL statement to find the average class scores for all courses, grouped by the area over the different years. The query should return 3 columns: year, area, avg_score. Results should be ordered by year (ascending), then area (AZ).
3. If classes can belong to more than one area (e.g., it can be both a theory and systems class), what changes to the database schema are necessary?

Problem 3: Sparse Matrix Multiplication

A sparse matrix is a matrix populated primarily with zeros as elements of the table. Systems designed to efficiently support sparse matrices look a lot like databases: The **(i, j)** entry of the matrix is represented as a record **(i, j ,value)** in the database. The benefit is that you only need one record for every nonzero element in the matrix. For example, the matrix:

0 2 0
1 0 0
0 0 3
0 0 0
can be represented as a table:
row # column #
value
0 1 2
1 0 1
2 2 3

so that instead of having to store 12 integers, we only need to store 9. Now, since you can represent a sparse matrix as a table, it's reasonable to consider whether you can express matrix multiplication as a SQL query.
In matrix.db , there are two matrices **A** and **B** represented as follows:

- A(**row_num** int, **col_num** int, **value** int)
- B(**row_num** int, **col_num** int, **value** int)

The matrix **A** and matrix **B** are both square matrices with 5 rows and 5 columns. Express A x B as a SQL query. The query should return 3 columns: row_num, col_num, value. Results should be ordered by row_num (ascending), then col_num (ascending). If you're wondering why this might be a good idea, consider that advanced databases execute queries in parallel. So it can be quite efficient to process a very large sparse matrix in a database (millions of rows and columns)!