Analytics Lab: Use a Relational Database for University Data

Solutions to the Technical Challenges

(Solutions to challenges 1-4 are not provided. For help setting up your database, reach out to your classmates and instructor.)

5. Try out some queries, and briefly explain what they do and how many records you see in the results as comments for each query below:

```sql
SELECT * FROM instructor;
```
12 records

```sql
SELECT name FROM instructor
WHERE dept_name = 'Comp. Sci.' AND salary >
70000;
```
2 records

```sql
SELECT * FROM instructor, department
WHERE instructor.dept_name =
department.dept_name;
```
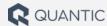**12 records**

6. Write a SQL query to find the names of all the instructors from the Biology department.

```sql
SELECT name
FROM instructor
WHERE dept_name = 'Biology';
```

QUANTIC

7. Write a SQL query to find the names of courses in the Computer Science department which have 3 credits.

```sql
SELECT title
FROM course
WHERE dept_name = 'Computer Science' AND credits
= 3;
```

8. For the student with ID 12345, write a SQL query to show the course_id and title of all courses registered for by the student.
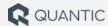
```sql
SELECT t.course_id, c.title
FROM takes AS t
JOIN course AS c ON t.course_id = c.course_id
WHERE t.ID = '12345';
```

9. Write a SQL query to display the IDs of all instructors who have never taught a course.

```sql
SELECT ID
FROM instructor AS i
WHERE NOT EXISTS (
      SELECT *
      FROM teaches AS t
      WHERE i.ID = t.ID
);
```

10. Write a SQL query to find the names of all students who have ever taken any Comp. Sci. course. (There should be no duplicate names.)

```sql
SELECT DISTINCT s.name
FROM student AS s
JOIN takes AS t ON s.ID = t.ID
JOIN course AS c ON t.course_id = c.course_id
WHERE c.dept_name = 'Comp. Sci.';
```

11. Write a SQL query to find the maximum and minimum enrollment across all sections. Consider only sections that had some enrollment; don't worry about sections with no enrolled students.

```sql
SELECT MAX(enrollment_count) AS max_enrollment,
MIN(enrollment_count) AS min_enrollment
FROM (
      SELECT course_id, sec_id, semester, year,
COUNT(*) AS enrollment_count
      FROM takes
      GROUP BY course_id, sec_id, semester, year
) AS section_enrollments;
```

12. Write a SQL query to return the section(s) with the highest enrollment for all courses, and list the enrollment for those sections, using a subquery.

```sql
SELECT course_id, sec_id, semester, year,
enrollment_count
FROM (
      SELECT course_id, sec_id, semester, year,
COUNT(*) AS enrollment_count,
            RANK() OVER (PARTITION BY course_id
ORDER BY COUNT(*) DESC) AS rank
      FROM takes
      GROUP BY course_id, sec_id, semester, year
) AS section_enrollments
WHERE rank = 1;
```

13. Grades are mapped to a grade point as follows: A:10, B:8, C:6, D:4 and F:0. Create a table to store these mappings, and use it to write a query to find the cumulative grade point average (GPA).

    a. GPA is calculated by averaging the grade points earned by a student in all courses. For example, if a student earns an A (4.0), a B (3.0), and a C (2.0) in three different courses, the average would be (4.0 + 3.0 + 2.0) / 3 = 3.0, resulting in a cumulative GPA of 3.0.

    b. To ensure that no student receives a GPA before they have received all their final grades, make sure that, for students who have a null grade in any course they have taken, the cumulative GPA is shown as null.

```sql
 --Create grade_points table
CREATE TABLE grade_points (
      grade CHAR(1) PRIMARY KEY,
      grade_point INTEGER
);

-- Insert the grade mappings
INSERT INTO grade_points (grade, grade_point) VALUES
('A', 10),
('B', 8),
('C', 6),
('D', 4),
('F', 0);

SELECT t.ID,
      CASE
            WHEN COUNT(t.grade) FILTER (WHERE t.grade
IS NULL) > 0 THEN NULL
            ELSE AVG(gp.grade_point)
      END AS cumulative_gpa
FROM takes AS t
LEFT JOIN grade_points AS gp ON t.grade = gp.grade
GROUP BY t.ID;
```