29702361—SQL Programming & Creative Writing

Task 1

- Considering the attached `Student` and `Student_Courses` relations, discuss when Union, Inner Join, Left Outer Join, Right Outer Join, and Full Outer Join should be used and provide examples using the attached relations.

> *Cite any source in APA format.*

`UNION`

The `UNION` operations enables queries to combine their outputs if the provided attributes contain similar data types and attribute names (Elmasri & Navathe, 2016). A case in point is where you want to list the `Student_ID` attributes from both `Student` and `Student_Courses`. Using MySQL, for instance, that could be achieved by executing the query:

```
SELECT Student_ID FROM Student
UNION
SELECT Student_ID FROM Student_Courses;
```

This would produce the output:

```
+------------+
| Student_ID |
+------------+
| 0000       |
| 1111       |
| 2222       |
| 3333       |
| 4444       |
| 5555       |
+------------+
6 rows in set (0.001 sec)
```

This lists the distinct rows containing the `Student_ID` values. On the other hand, by tweaking the earlier query to use the `UNION ALL` operation we get the output:

```
+------------+
| Student_ID |
+------------+
| 0000       |
| 1111       |
| 2222       |
| 3333       |
| 4444       |
| 5555       |
| 2222       |
| 2222       |
| 2222       |
| 0000       |
| 1111       |
| 1111       |
| 0000       |
| 1111       |
| 1111       |
| 3333       |
| 3333       |
| 4444       |
| 4444       |
| 4444       |
+------------+
20 rows in set (0.001 sec)
```

INNER JOIN

The INNER JOIN operation lists all the rows from the specified tables that meet the a given criteria (Elmasri & Navathe, 2016). As an example, if you want to select all the rows from with Student_Name, Dept_Name, and Course_ID attributes from the Student and Student_Courses relations; where, the Student_ID values match in both relations and the Grade attribute in Student_Courses has an A value; you would write such a query as:

```
SELECT Student_Name, Dept_Name, Course_ID
FROM Student
INNER JOIN Student_Courses
ON Student.Student_ID = Student_Courses.Student_ID
AND Student_Courses.Grade = 'A';
```

Which would then produce the output of:

```
+--------------+------------------+-----------+
| Student_Name | Dept_Name        | Course_ID |
+--------------+------------------+-----------+
| AAAAA        | Computer Science | CS 1101   |
| AAAAA        | Computer Science | CS 2203   |
| BBBBB        | Computer Science | CS 2204   |
| BBBBB        | Computer Science | CS 2401   |
| DDDDD        | Education        | EDUC 5010 |
| EEEEE        | Physics          | PHY 1101  |
| EEEEE        | Physics          | PHY 3304  |
+--------------+------------------+-----------+
7 rows in set (0.001 sec)
```

LEFT OUTER JOIN

The LEFT OUTER JOIN operation "returns not only the rows matching the join condition (that is, rows with matching values in the common column), but also the rows in the left side table with unmatched values in the right side table" (Rob & Coronel, 2009 , pg. 310). As an example, say you want to list all the available Student relations using their Student_ID and Student_Name attributes together with the courses that they do, you would write a query such as:

```
SELECT Student.Student_ID, Student_Name, Course_ID
FROM Student
LEFT OUTER JOIN Student_Courses
ON Student.Student_ID = Student_Courses.Student_ID;
```

On execution, you will notice that the Student named FFFFF has not enrolled in any course:

```
+------------+--------------+-----------+
| Student_ID | Student_Name | Course_ID |
+------------+--------------+-----------+
| 0000       | AAAAA        | CS 1101   |
| 0000       | AAAAA        | CS 2203   |
| 1111       | BBBBB        | CS 1101   |
| 1111       | BBBBB        | CS 1102   |
| 1111       | BBBBB        | CS 2204   |
| 1111       | BBBBB        | CS 2401   |
| 2222       | CCCCC        | BUS 1101  |
| 2222       | CCCCC        | BUS 2201  |
| 2222       | CCCCC        | BUS 3302  |
| 3333       | DDDDD        | EDUC 5010 |
| 3333       | DDDDD        | EDUC 5210 |
| 4444       | EEEEE        | PHY 1101  |
| 4444       | EEEEE        | PHY 2202  |
| 4444       | EEEEE        | PHY 3304  |
| 5555       | FFFFF        | NULL      |
+------------+--------------+-----------+
15 rows in set (0.001 sec)
```

Additionally, because the left table in this case is the Student relation, the ordering of the Student_ID attributes in the Student relation define how the rest of the columns are sorted.

`RIGHT OUTER JOIN`

The `RIGHT OUTER JOIN` operation "returns not only the rows matching the join condition (that is, rows with matching values in the common column), but also the rows in the right side table with unmatched values in the left side table" (Rob & Coronel, 2009 , pg. 310). As an example, say you want to list all only those courses that students have took in a table containing the `Student_ID`, `Student_Name`, and `Course_ID` relations, you would write a query such as:

```sql
SELECT Student.Student_ID, Student_Name, Course_ID
FROM Student
RIGHT OUTER JOIN Student_Courses
ON Student.Student_ID = Student_Courses.Student_ID;
```

Also, this output shows how one can eliminate the `NULL` values from the previous section:

```
+------------+--------------+-----------+
| Student_ID | Student_Name | Course_ID |
+------------+--------------+-----------+
| 2222       | CCCCC        | BUS 1101  |
| 2222       | CCCCC        | BUS 2201  |
| 2222       | CCCCC        | BUS 3302  |
| 0000       | AAAAA        | CS 1101   |
| 1111       | BBBBB        | CS 1101   |
| 1111       | BBBBB        | CS 1102   |
| 0000       | AAAAA        | CS 2203   |
| 1111       | BBBBB        | CS 2204   |
| 1111       | BBBBB        | CS 2401   |
| 3333       | DDDDD        | EDUC 5010 |
| 3333       | DDDDD        | EDUC 5210 |
| 4444       | EEEEE        | PHY 1101  |
| 4444       | EEEEE        | PHY 2202  |
| 4444       | EEEEE        | PHY 3304  |
+------------+--------------+-----------+
14 rows in set (0.001 sec)
```

`FULL OUTER JOIN`

The `FULL OUTER JOIN` operation "returns not only the rows matching the join condition (that is, rows with matching values in the common column), but also all of the rows with unmatched values in either side table" (Rob & Coronel, 2009 , pg. 311). In MySQL, a `FULL OUTER JOIN` operation can be realized by combining the outputs of the `LEFT OUTER JOIN` and `RIGHT OUTER JOIN` operations using a `UNION` operator—as this query shows:

```sql
SELECT Student.Student_ID, Student_Name, Course_ID
FROM Student
LEFT OUTER JOIN Student_Courses
ON Student.Student_ID = Student_Courses.Student_ID
UNION
SELECT Student.Student_ID, Student_Name, Course_ID
FROM Student
RIGHT OUTER JOIN Student_Courses
ON Student.Student_ID = Student_Courses.Student_ID;
```

The resultant output:

```
+------------+--------------+-----------+
| Student_ID | Student_Name | Course_ID |
+------------+--------------+-----------+
| 0000       | AAAAA        | CS 1101   |
| 0000       | AAAAA        | CS 2203   |
| 1111       | BBBBB        | CS 1101   |
| 1111       | BBBBB        | CS 1102   |
| 1111       | BBBBB        | CS 2204   |
| 1111       | BBBBB        | CS 2401   |
| 2222       | CCCCC        | BUS 1101  |
| 2222       | CCCCC        | BUS 2201  |
| 2222       | CCCCC        | BUS 3302  |
| 3333       | DDDDD        | EDUC 5010 |
| 3333       | DDDDD        | EDUC 5210 |
| 4444       | EEEEE        | PHY 1101  |
| 4444       | EEEEE        | PHY 2202  |
| 4444       | EEEEE        | PHY 3304  |
| 5555       | FFFFF        | NULL      |
+------------+--------------+-----------+
15 rows in set (0.001 sec)
```

References

Elmasri, R., & Navathe, S. (2016). *Fundamentals of database systems*. Pearson.

Rob, P., & Coronel, C. (2009). *Database systems: Design, implementation, and management*. Thomson/Course Technology.

Task 2

- Create two different “Views” using the attached relations, then Alter one of the Views and Drop the other one. After that, discuss why Views are used by giving examples using the attached relations.
- Describe what you did (This does not mean that you copy and paste from what you have posted or the assignments you have prepared. You need to describe what you did and how you did it), what you learned, your weekly activities, in what ways are you able to apply the ideas and concepts gained, and finally, describe one important thing that you are thinking about in relation to the activity.

> *Cite any source in APA format.*

In a case where one wants to be able to list the details of a `Student` relation using only the `Student_ID` and `Student_Name` attributes, it would be convenient to create a View that executes a given `SELECT` on the underlying table instead of having to run that query repeatedly by oneself. Thus to create such a View (named `student_details`), one would create it using the DDL command:

```
CREATE VIEW student_details AS
    SELECT Student_ID, Student_Name
    FROM Student;
```

Accordingly, to use the View, it is as simple as running the query:

```
SELECT * FROM student_details;
```

Which produces the output:

```
+------------+--------------+
| Student_ID | Student_Name |
+------------+--------------+
| 0000       | AAAAA        |
| 1111       | BBBBB        |
| 2222       | CCCCC        |
| 3333       | DDDDD        |
| 4444       | EEEEE        |
| 5555       | FFFFF        |
+------------+--------------+
6 rows in set (0.001 sec)
```

Yet, this View does not offer enough detail when one wants to know what marks a student scored in a course, example. Thus, it is necessary to create another View named `student_marks`, which contains both the identifying details of the students and their course grades:

```
CREATE VIEW student_marks AS
SELECT Student_Name, Dept_Name, Course_ID, Year, Grade FROM Student
LEFT OUTER JOIN Student_Courses ON
Student.Student_ID = Student_Courses.Student_ID;
```

On querying the `student_marks` View:

```
SELECT * FROM student_marks;
```

The resulting output is:

```
+--------------+-------------------------+-----------+------+-------+
| Student_Name | Dept_Name               | Course_ID | Year | Grade |
+--------------+-------------------------+-----------+------+-------+
| AAAAA        | Computer Science        | CS 1101   | 2019 | A     |
| AAAAA        | Computer Science        | CS 2203   | 2020 | A     |
| BBBBB        | Computer Science        | CS 1101   | 2019 | B     |
| BBBBB        | Computer Science        | CS 1102   | 2019 | B     |
| BBBBB        | Computer Science        | CS 2204   | 2020 | A     |
| BBBBB        | Computer Science        | CS 2401   | 2020 | A     |
| CCCCC        | Business Administration | BUS 1101  | 2019 | C     |
| CCCCC        | Business Administration | BUS 2201  | 2020 | B     |
| CCCCC        | Business Administration | BUS 3302  | 2020 | B     |
| DDDDD        | Education               | EDUC 5010 | 2019 | A     |
| DDDDD        | Education               | EDUC 5210 | 2020 | B     |
| EEEEE        | Physics                 | PHY 1101  | 2019 | A     |
| EEEEE        | Physics                 | PHY 2202  | 2020 | C     |
| EEEEE        | Physics                 | PHY 3304  | 2020 | A     |
| FFFFF        | History                 | NULL      | NULL | NULL  |
+--------------+-------------------------+-----------+------+-------+
15 rows in set (0.001 sec)
```

Still, the `student_marks` View is deficient in one aspect. It does not list the `Student_ID` attributes. Hence, it requires some modification, which can be achieved by altering it using the following DDL command:

```
ALTER VIEW student_marks AS
SELECT Student.Student_ID, Student_Name, Dept_Name, Course_ID, Year, Grade FROM Student
LEFT OUTER JOIN Student_Courses ON
Student.Student_ID = Student_Courses.Student_ID;
```

On querying the altered `student_marks` View:

```
SELECT * FROM student_marks;
```

The resulting output shows that a new column named `Student_ID` has been created:

```
+------------+--------------+-------------------------+-----------+------+-------+
| Student_ID | Student_Name | Dept_Name               | Course_ID | Year | Grade |
+------------+--------------+-------------------------+-----------+------+-------+
| 0000       | AAAAA        | Computer Science        | CS 1101   | 2019 | A     |
| 0000       | AAAAA        | Computer Science        | CS 2203   | 2020 | A     |
| 1111       | BBBBB        | Computer Science        | CS 1101   | 2019 | B     |
| 1111       | BBBBB        | Computer Science        | CS 1102   | 2019 | B     |
| 1111       | BBBBB        | Computer Science        | CS 2204   | 2020 | A     |
| 1111       | BBBBB        | Computer Science        | CS 2401   | 2020 | A     |
| 2222       | CCCCC        | Business Administration | BUS 1101  | 2019 | C     |
| 2222       | CCCCC        | Business Administration | BUS 2201  | 2020 | B     |
| 2222       | CCCCC        | Business Administration | BUS 3302  | 2020 | B     |
| 3333       | DDDDD        | Education               | EDUC 5010 | 2019 | A     |
| 3333       | DDDDD        | Education               | EDUC 5210 | 2020 | B     |
| 4444       | EEEEE        | Physics                 | PHY 1101  | 2019 | A     |
| 4444       | EEEEE        | Physics                 | PHY 2202  | 2020 | C     |
| 4444       | EEEEE        | Physics                 | PHY 3304  | 2020 | A     |
| 5555       | FFFFF        | History                 | NULL      | NULL | NULL  |
+------------+--------------+-------------------------+-----------+------+-------+
15 rows in set (0.001 sec)
```

Finally, because this `student_marks` View contains all the info we need on a student's performance—including the identifying details, the `student_details` View is rendered redundant and removing it would be welcome. This is attainable using the `DROP VIEW` operation:

```
DROP VIEW student_details;
```

As a result, any further use of that View identifier is bound to throw an error.

The MySQL installation contains a command line tool , which I used extensively when creating, altering, and dropping the views for this task. Also, I used it for executing `SELECT` statements that allowed me to inspect the contents of the views and their underlying tables. Thus when creating the `student_details` view, it was as straightforward as running its DDL command on the terminal. Yet, this was only the execution part of the task. First, I had to assess the provided relations to determine which attributes would be most helpful for a user who possesses the data of students and their course grades. Second, to complete the task, I had to experiment with various types of views to come up with those that combined the most insightful data from both the provided relations. All in all, the testing would not have been complete if I had not created actual database tables containing the attributes and values displayed in the provided relation screen-shots.

All these activities reminded my how important it was to know how to generate DDL statements that could generate tables and data matching given use cases. Thus, I was pleased by how adept I have become at handling SQL tasks because I could borrow easily from skills learned in earlier SQL assignments. On the other hand, I learned that views only represent a virtual form of the tables that a database system contains. It was actually a relieving surprising to find out that I could create, alter, and even drop views without affecting the data that was held by the tables in the database. This was especially crucial because at some point I had hesitated to implement the `DROP VIEW` routines out of fear that I would delete all the data that I had entered into the tables manually.