SQL Queries

- **SELECT (σ) :**
  - The SELECT operation is used for **selecting a subset of the tuples according to a given selection condition**.
  - Sigma(σ)Symbol denotes it.
  - It is used as an expression to choose tuples which meet the selection condition.
  - Select operator selects tuples that satisfy a given predicate.

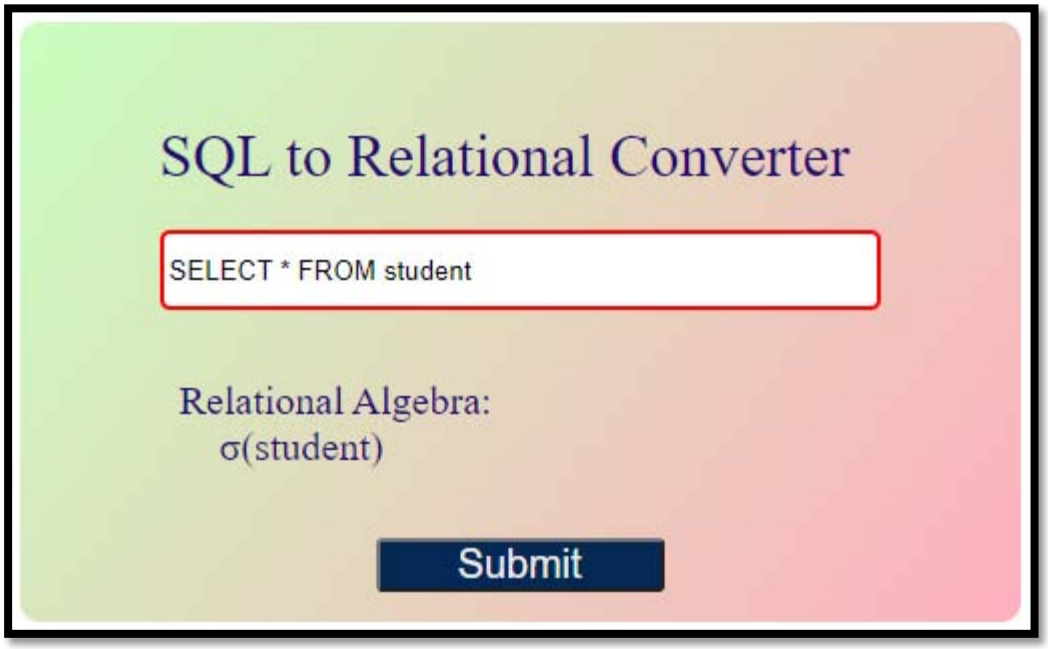
Code for SELECT operation:

```
 const SELECT = "&#963"

 function selectExpression(expression) {
remove = /(?<=((?:order by)|(?:group by)|(?:having)))(.*)/

rem_ans = remove.exec(expression)

if (rem_ans != null) {
    expression = expression.replace(rem_ans[1] + rem_ans[2], "").trim();
}

const re = /select ([a-z ,\.]+|\*) from ([a-z0-9]+)[ ]?(.*)/

if ((myarray = re.exec(expression)) != null) {
    answer = SELECT

    if (myarray[3] != null) {
        answer += '<sub>' + getPredicate(myarray[3]) + '</sub>'
    }

    answer += '(' + myarray[2] + ')'

    if (myarray[1] == "*") {
        return answer;
    }
    myarray[1] = myarray[1].replaceAll(" as ", "/")
    return PROJECT + "<sub>" + myarray[1] + "</sub>" + "(" + answer + ")"
}

return expression
}
```

Output: SELECT Operation:

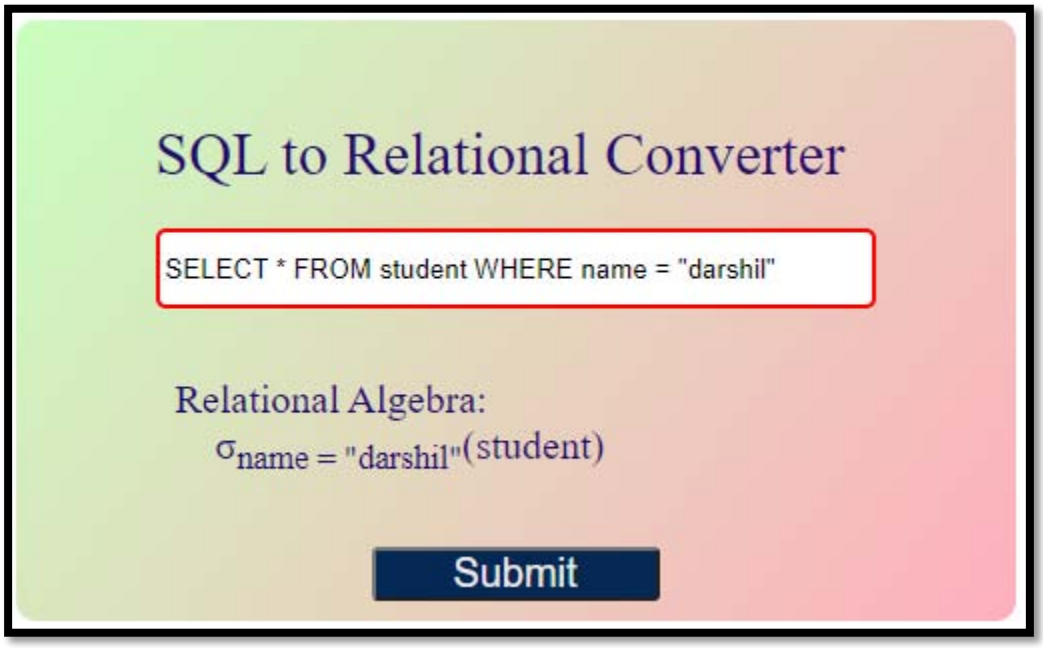
SQL Query:

- SELECT * FROM student



SQL Query:

- SELECT * FROM student WHERE name = "darshil"



- **PROJECT (π):**
  - Project operator is denoted by ∏ symbol and it is used **to select desired columns (or attributes) from a table (or relation)**. Project operator in relational algebra is similar to the Select statement in SQL.

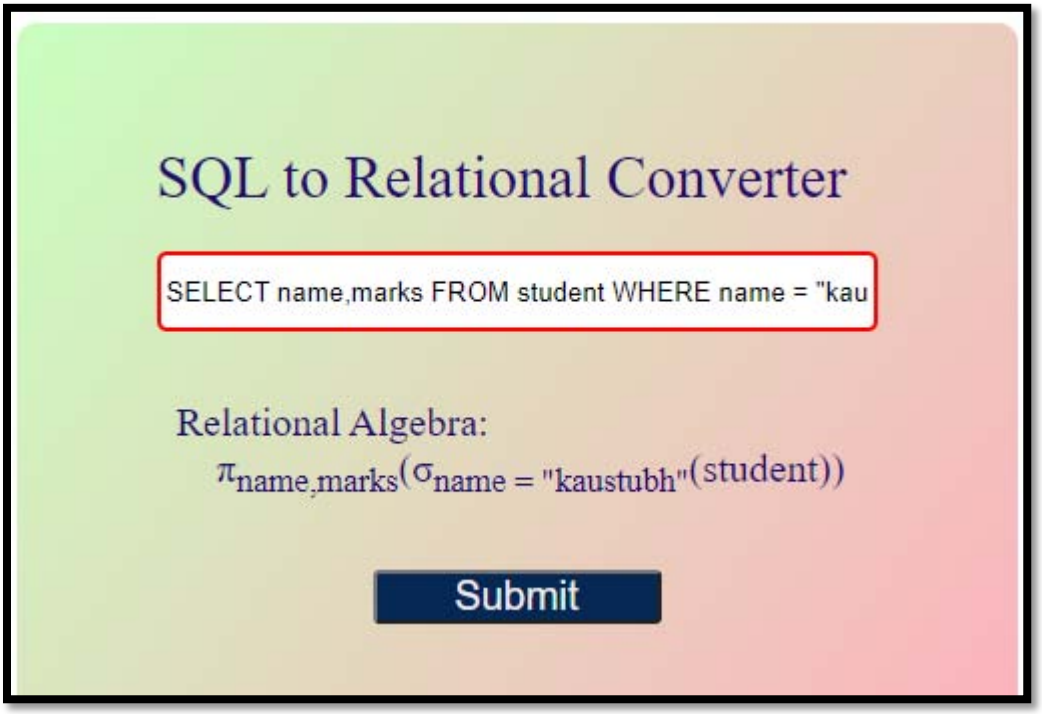
Code for Project Operation:

```
const PROJECT = "&#960"

function selectExpression(expression) {
remove = /(?<=((?:order by)|(?:group by)|(?:having)))(.*)/

rem_ans = remove.exec(expression)

if (rem_ans != null) {
    expression = expression.replace(rem_ans[1] + rem_ans[2], "").trim();
}

const re = /select ([a-z ,\.]+|\*) from ([a-z0-9]+)[ ]?(.*)/

if ((myarray = re.exec(expression)) != null) {
    answer = SELECT

    if (myarray[3] != null) {
        answer += '<sub>' + getPredicate(myarray[3]) + '</sub>'
    }

    answer += '(' + myarray[2] + ')'

    if (myarray[1] == "*") {
        return answer;
    }
    myarray[1] = myarray[1].replaceAll(" as ", "/")
    return PROJECT + "<sub>" + myarray[1] + "</sub>" + "(" + answer + ")"
}

return expression
}
```

Output - PROJECT operation:

SQL Query:

- SELECTname, marks FROM student WHERE name = "kaustubh"



- **AND (∧) , OR (∨):**
  - The AND, and OR operator is used **to combine multiple conditions in an SQL statement's WHERE clause**.

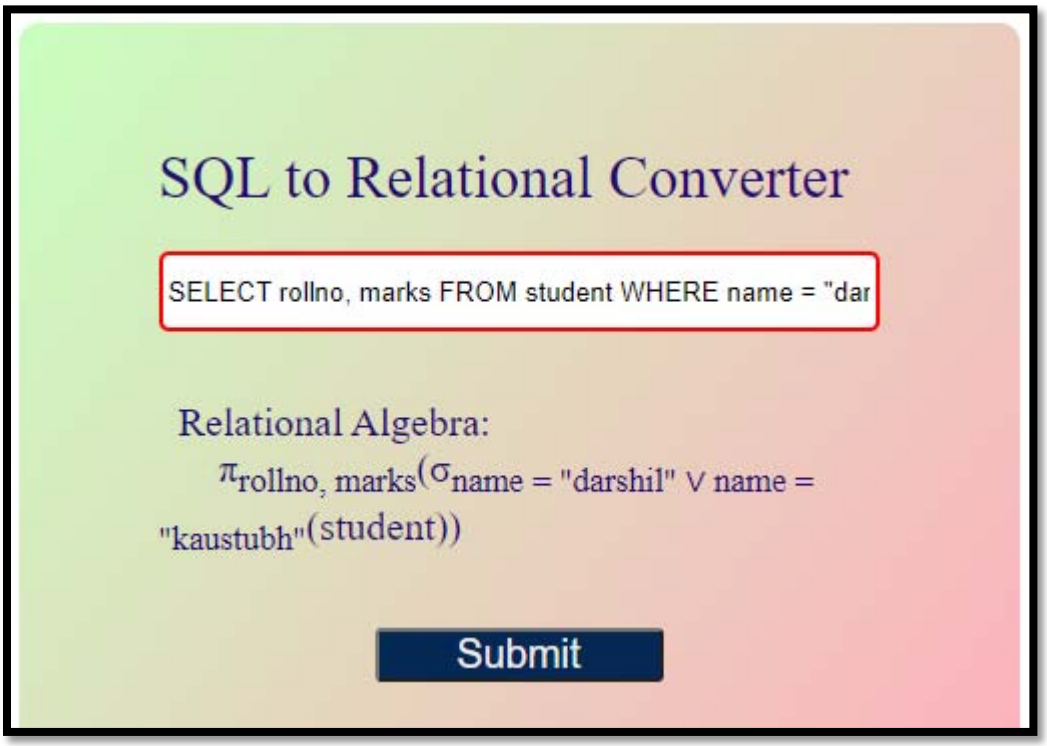
Code for AND and OR operation:

```
 const OR = " &#8744 "
 const AND = " &#8743 "

 function getPredicate(expression) {
const re = /where (.*)[ ]?\(?.*/

conditions = re.exec(expression);

if (conditions == null) {
   return "";
}

modifiedConditions = conditions[1].replace(" and ", AND)
modifiedConditions = modifiedConditions.replace(" or ", OR)

return modifiedConditions
}
```

Output:OR operation:

SQL Query:

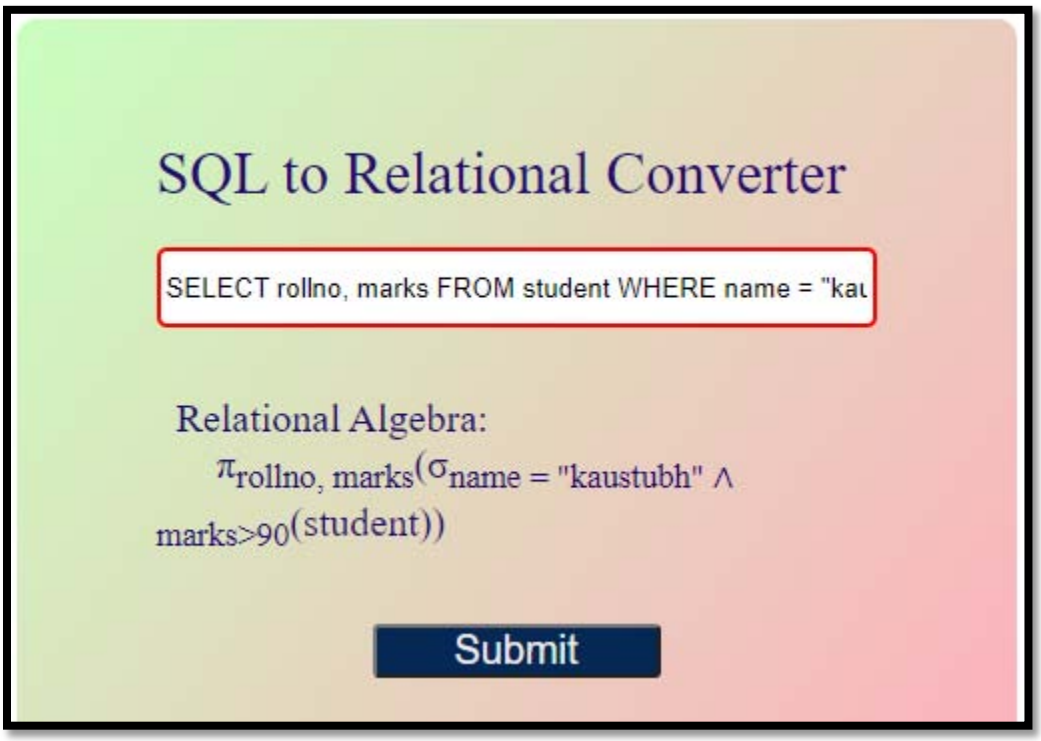
- SELECT rollno, marks FROM student WHERE name = "darshil" or name = "kaustubh"



Output: AND operation:

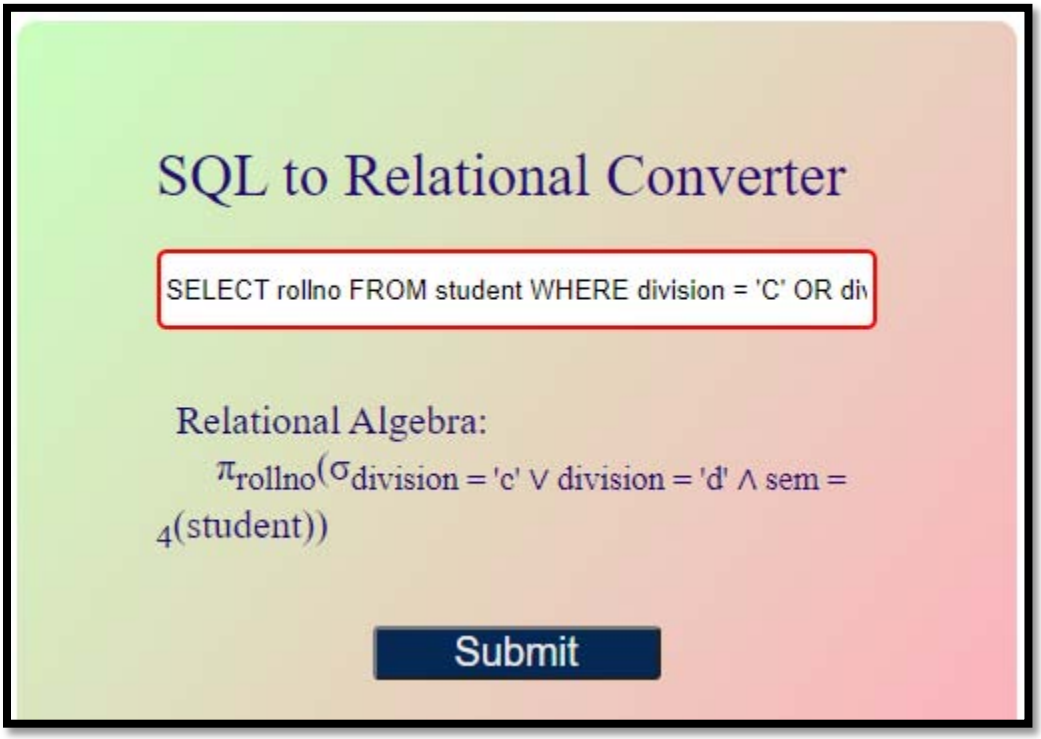
SQL Query:

- SELECT rollno, marks FROM student WHERE name = "kaustubh" and marks > 90



SQL Query:

- SELECT rollno FROM student WHERE division = 'C' OR division = 'D' AND sem = 4



- **MINUS (–):**
  - The SQL MINUS operator is used **to return all rows in the first SELECT statement that are not returned by the second SELECT statement**.
  - The MINUS operator will retrieve all records from the first dataset and then remove from the results all records from the second dataset.

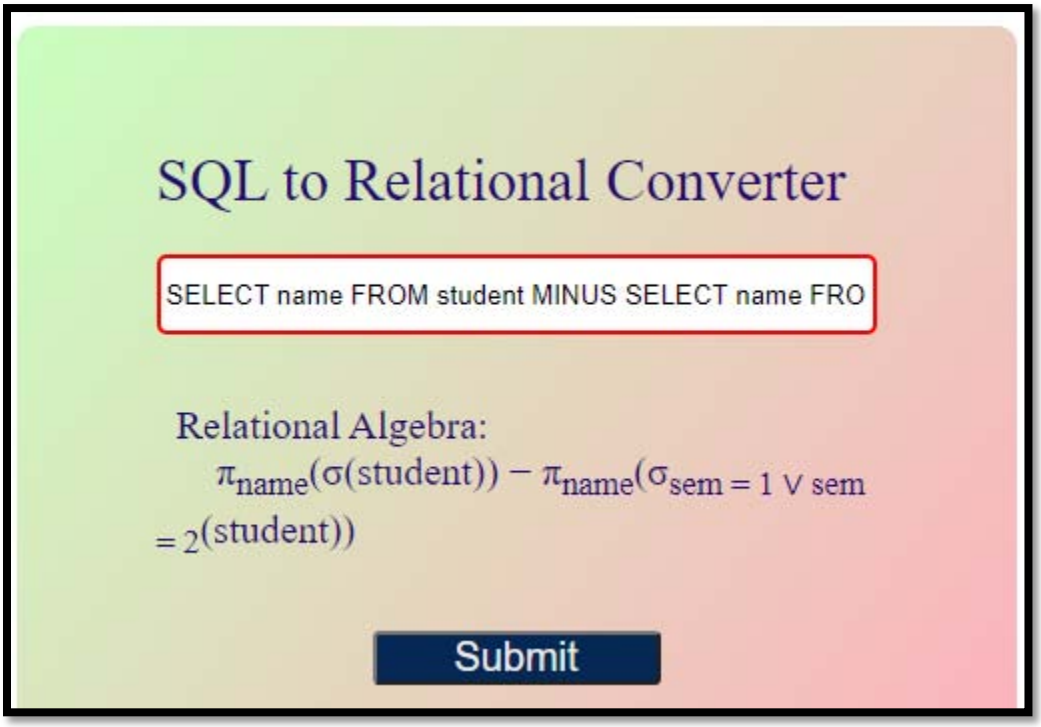
Code for MINUS operaion:

```
const MINUS = " &#8722 "

function minusOperation(expression) {
const minus = /(.*) minus (.*)/

query = minus.exec(expression)

if (query != null) {
    return selectExpression(query[1]) + MINUS + selectExpression(query[2])
}

return ""
}
```

Output: MINUS operation:

SQL Query:

- SELECT name FROM student MINUS SELECT name FROM student WHERE sem = 1 OR sem = 2



- **INTERSECT (∩):**
  - The SQL INTERSECT clause/operator is used **to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement**.
  - This means INTERSECT returns only common rows returned by the two SELECT statements.

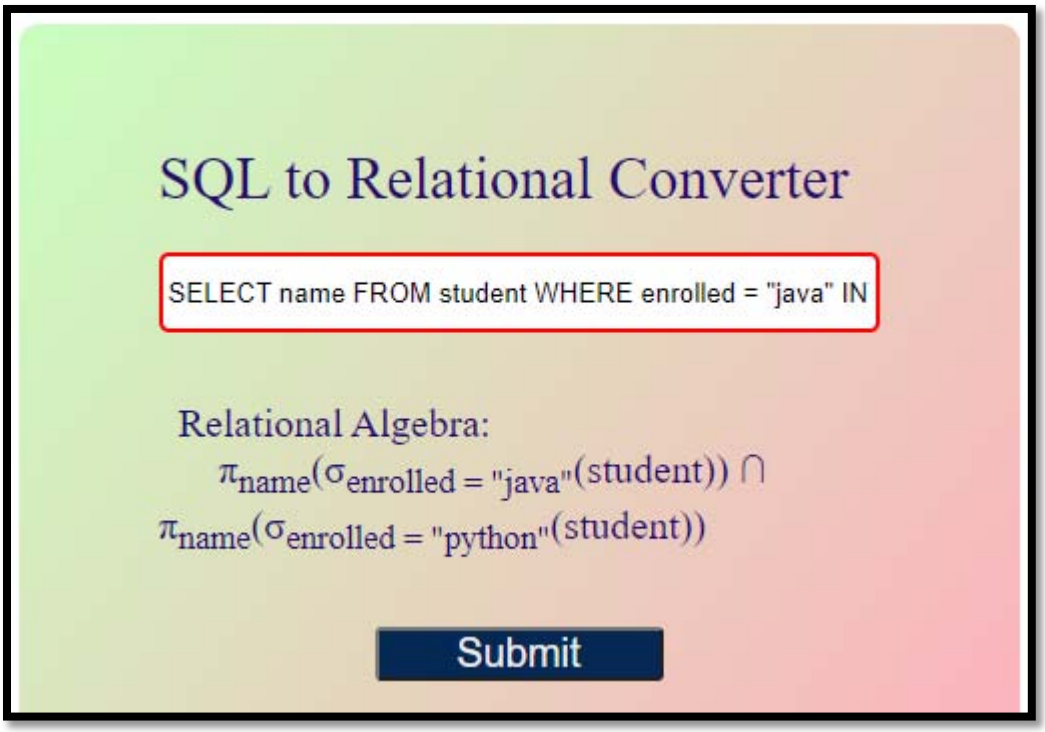
Code for INTERSECT operation:

```
 const INTERSECT = " &#8745 "

 function intersectOperation(expression) {
const intersect = /(.*) intersect (.*)/

query = intersect.exec(expression)

if (query != null) {
    return selectExpression(query[1]) + INTERSECT + selectExpression(query[2])
}

return ""
}
```

Output: INTERSECT operation:

SQL Query:

- SELECT name FROM student WHERE enrolled = "java" INTERSECT SELECT name FROM student WHERE enrolled = "python"



- **UNION (U):**
  - The SQL Union operation is used **to combine the result of two or more SQL SELECT queries**.
  - In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
  - The union operation eliminates the duplicate rows from its resultset.

Code for UNION operation:

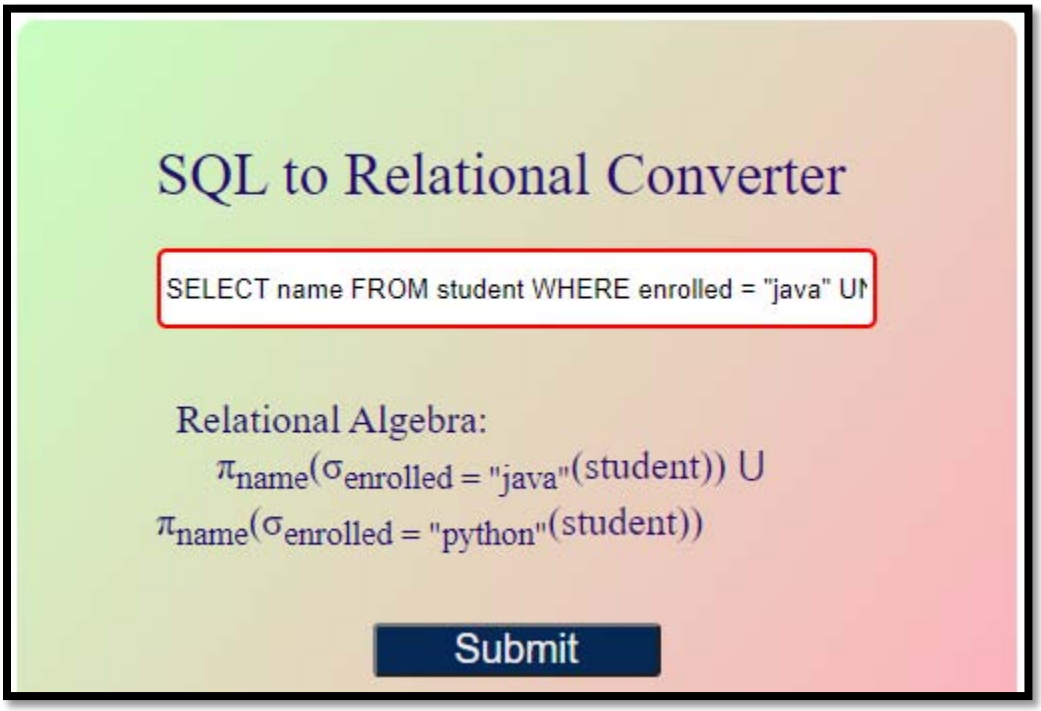
```
const UNION = " &#8899 "

function unionOperation(expression) {
const union = /(.*) union (.*)/

query = union.exec(expression)

if (query != null) {
   return selectExpression(query[1]) + UNION + selectExpression(query[2])
}
```

```
    return ""
}
```

Output: UNION operation:

SQL Query:

- SELECT name FROM student WHERE enrolled = "java" UNION SELECT name FROM student WHERE enrolled = "python"



- **CROSS JOIN (×):**
  - The CROSS JOIN is used **to generate a paired combination of each row of the first table with each row of the second table**.
  - This join type is also known as cartesian join.

Code for CROSS JOIN:

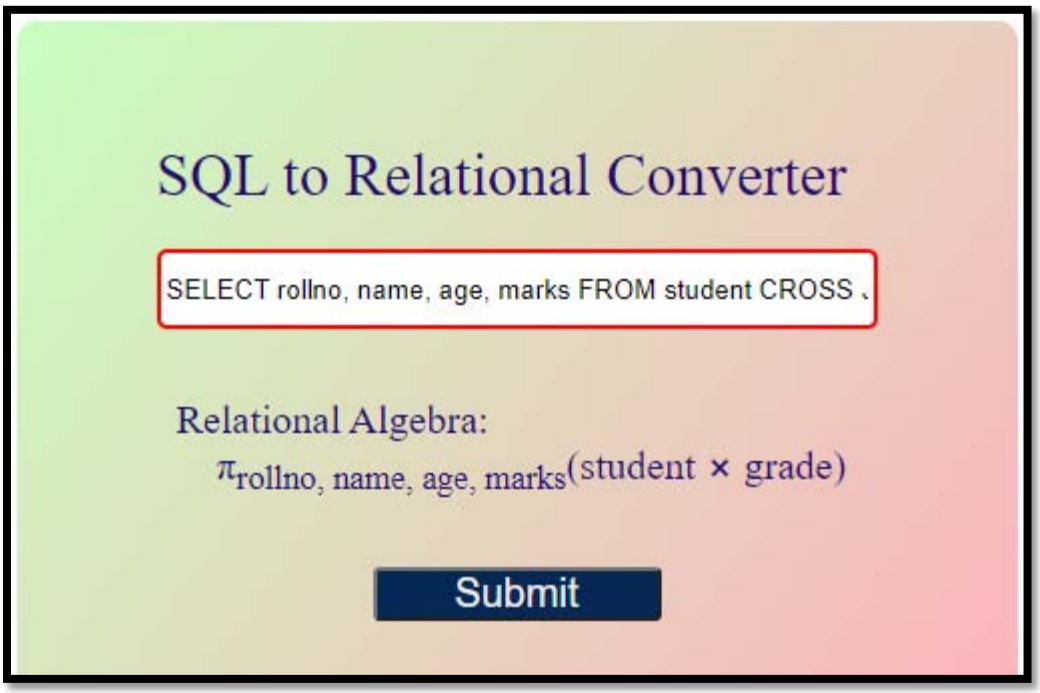
```
 const CROSS_JOIN = " &#10799 "

 function crossJoinOperation(expression) {
const crossJoin = /select (.*) from ([a-zA-Z0-9_]+) cross join ([a-zA-Z0-9_]+)[ ]?(.*)/

query = crossJoin.exec(expression)

if (query == null) {
```

```
            return ""
        }

        if (query[4] == null) {
            value = (query[1] == "*") ? "" : query[1].replaceAll(" as ", "/")
            return PROJECT + "<sub>" + value + "</sub>" + "(" + query[2] + CROSS_JOIN + query[3] +
")"
        } else {
            value = (query[1] == "*") ? "" : query[1].replaceAll(" as ", "/")
            match = /where (.*)/g
            ans = match.exec(query[4])

            if (ans == null) {
                return PROJECT + "<sub>" + value + "</sub>" + "(" + query[2] + CROSS_JOIN + query[3]
+ ")"
            } else {
                a = ans[1].replaceAll(" and ", AND)
                a = a.replaceAll(" or ", OR)

                return PROJECT + "<sub>" + value + "</sub>" + "(" + SELECT + "<sub>" + a + "</sub>" +
"(" + query[2] + CROSS_JOIN + query[3] + "))"
            }
        }
    }
```

Output: CROSS JOIN:

SQL Query:

- SELECT rollno, name, age, marks FROM student CROSS JOIN grade



- **NATURAL JOIN(⋈):**
  - A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.

Code for NATURAL JOIN:

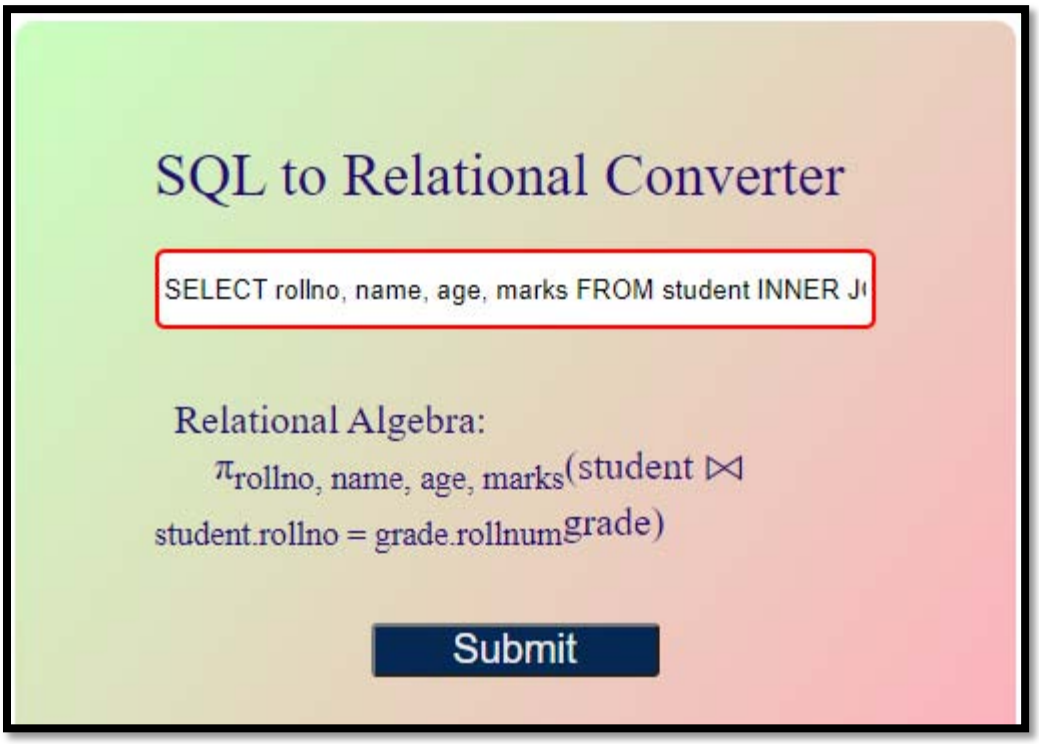
```
function naturalJoinOperation(expression) {

const naturalJoin = /select (.*) from ([a-zA-Z0-9_]+) natural join ([a-zA-Z0-9_]+).*(where .*)?/

query = naturalJoin.exec(expression)

if (query == null) {
    return ""
}

if (query[4] == null) {
    value = (query[1] == "*") ? "" : query[1].replaceAll(" as ", "/")
    return PROJECT + "<sub>" + value + "</sub>" + "(" + query[2] + THETA_JOIN + query[3] +
")"
} else {
    value = (query[1] == "*") ? "" : query[1].replaceAll(" as ", "/")
    match = /where (.*)/g
    ans = match.exec(query[4])

    if (ans == null) {
```

```
                return PROJECT + "<sub>" + value + "</sub>" + "(" + query[2] + THETA_JOIN + query[3]
+ ")"
            } else {
                a = ans[1].replaceAll(" and ", AND)
                a = a.replaceAll(" or ", OR)

                return PROJECT + "<sub>" + value + "</sub>" + "(" + SELECT + "<sub>" + a + "</sub>" +
"(" + query[2] + THETA_JOIN + query[3] + "))"
            }
        }
    }
```

Output: NATURAL JOIN :

SQL Query:

- SELECT rollno, name, age, marks FROM student NATURAL JOIN grade



- **LEFT OUTER JOIN(⟕):**
  - Left Outer Join returns all the rows from the table on the left and columns of the table on the right is null padded.
  - Left Outer Join retrieves all the rows from both the tables that satisfy the join condition along with the unmatched rows of the left table.

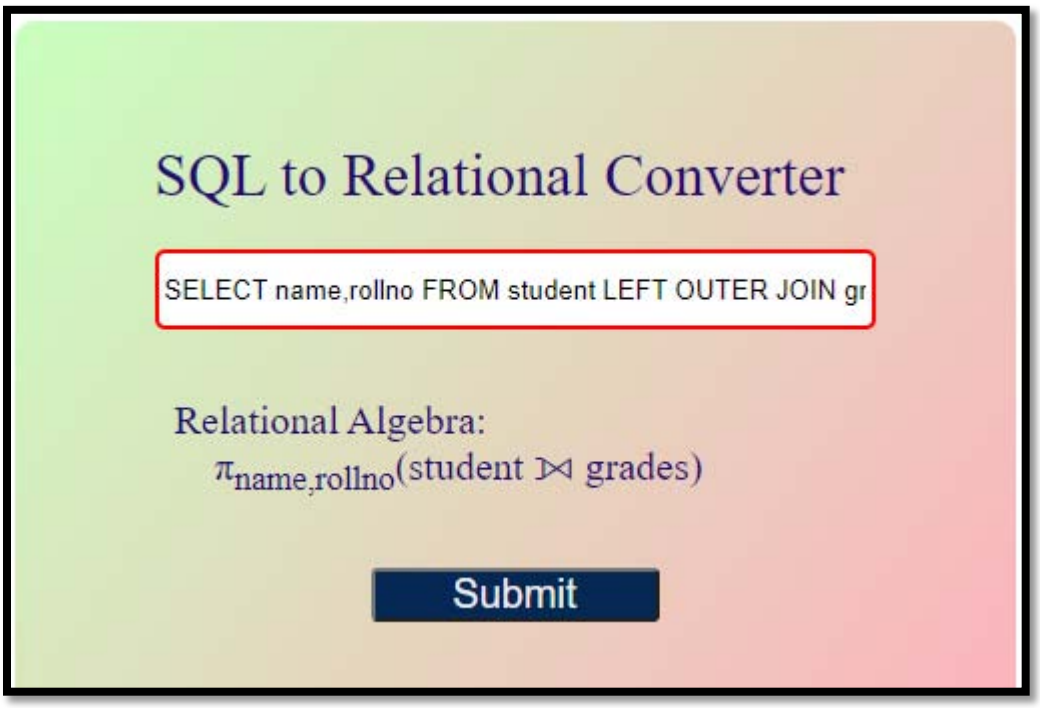
Code for LEFT OUTER JOIN:

```
        const LEFT_OUTER = " &#10197 "

        function leftOuterJoinOperation(expression) {

        const thetaJoin = /select (.*) from ([a-zA-Z0-9_]+) left outer join ([a-zA-Z0-9_]+) on (.*)[
]?(where .*)?/

        query = thetaJoin.exec(expression)

        if (query == null) {
            return ""
        }

        query[1] = (query[1] == "*") ? "" : query[1].replaceAll(" as ", "/")

        answer = PROJECT + "<sub>" + query[1] + "</sub>" + "("

        if (query[5] != null) {
            where = query[5].replace("where ", "")
            answer = answer + SELECT + "<sub>" + where + "</sub>" + "(" + query[2] + LEFT_OUTER
+ query[3] + "))"
        } else {
            answer = answer + query[2] + LEFT_OUTER + query[3] + ")"
        }

        return answer
    }
```

Output for LEFT OUTER JOIN:

SQL Query:

- SELECT name,rollno FROM student LEFTOUTER JOIN grades ON student.rno = grades.rno



- **RIGHT OUTER JOIN(⋊):**
  - Right Outer Join returns all the rows from the table on the right and columns of the table on the left is null padded.
  - Right Outer Join retrieves all the rows from both the tables that satisfy the join condition along with the unmatched rows of the right table.

Code for RIGHT OUTER JOIN:

```
const RIGHT_OUTER = " &#10198 "

function rightOuterJoinOperation(expression) {
const thetaJoin = /select (.*) from ([a-zA-Z0-9_]+) right outer join ([a-zA-Z0-9_]+) on (.*)[
]?(where .*)?/

query = thetaJoin.exec(expression)

if (query == null) {
    return ""
}

query[1] = (query[1] == "*") ? "" : query[1].replaceAll(" as ", "/")

answer = PROJECT + "<sub>" + query[1] + "</sub>" + "("

if (query[5] != null) {
    where = query[5].replace("where ", "")
```

```
            answer = answer + SELECT + "<sub>" + where + "</sub>" + "(" + query[2] +
RIGHT_OUTER + query[3] + "))"
        } else {
            answer = answer + query[2] + RIGHT_OUTER + query[3] + ")"
        }

        return answer
    }
```

Output for RIGHT OUTER JOIN:

SQL Query:

- SELECT name,rollno FROM student RIGHT OUTER JOIN grades ON student.rno = grades.rno