

# DBMS LAB

Name – RAVI SHEKHAR

Roll No. – 22CS3075

---

1.

```
class Relation:
```

```
    def __init__(self, attributes, tuples):
```

```
        self.attributes = attributes
```

```
        self.tuples = tuples
```

```
    def project(self, attributes_to_project):
```

```
        projected_tuples = []
```

```
        # Find the indices of attributes to project
```

```
        indices_to_project = [self.attributes.index(attr) for attr in attributes_to_project]
```

```
        # Project tuples
```

```
        for tuple in self.tuples:
```

```
            projected_tuple = [tuple[i] for i in indices_to_project]
```

```
            projected_tuples.append(projected_tuple)
```

```
        return Relation(attributes_to_project, projected_tuples)
```

```
    def __str__(self):
```

```
        header = " | ".join(self.attributes)
```

```
        lines = [header, "-" * len(header)]
```

```
        for tuple in self.tuples:
```

```
            lines.append(" | ".join(map(str, tuple)))
```

```
return "\n".join(lines)
```

```
# Example usage
```

```
if __name__ == "__main__":
```

```
    # Define the relation R
```

```
    R_attributes = ["Name", "Age", "City"]
```

```
    R_tuples = [
```

```
        ["Alice", 25, "New York"],
```

```
        ["Bob", 30, "Los Angeles"],
```

```
        ["Charlie", 35, "Chicago"]
```

```
    ]
```

```
    R = Relation(R_attributes, R_tuples)
```

```
    # Attributes to project on
```

```
    attributes_to_project = ["Name", "City"]
```

```
    # Perform projection
```

```
    projected_relation = R.project(attributes_to_project)
```

```
    # Output the resulting relation after the projection operation
```

```
    print("Original Relation R:")
```

```
    print(R)
```

```
    print("\nProjected Relation on", attributes_to_project, ":",)
```

```
    print(projected_relation)
```

The screenshot shows the Programiz Python Online Compiler interface. On the left, a file explorer shows 'main.py'. The main editor contains the following Python code:

```

1 class Relation:
2     def __init__(self, attributes, tuples):
3         self.attributes = attributes
4         self.tuples = tuples
5
6     def project(self, attributes_to_project):
7         projected_tuples = []
8         # Find the indices of attributes to project
9         indices_to_project = [self.attributes.index(attr) for attr in
10                                attributes_to_project]
11         # Project tuples
12         for tuple in self.tuples:
13             projected_tuple = [tuple[i] for i in indices_to_project]
14             projected_tuples.append(projected_tuple)
15         return Relation(attributes_to_project, projected_tuples)
16
17     def __str__(self):
18         header = " | ".join(self.attributes)
19         lines = [header, "-" * len(header)]
20         for tuple in self.tuples:
21             lines.append(" | ".join(map(str, tuple)))
22         return "\n".join(lines)
23
24 # Example usage
25 if __name__ == "__main__":
26     # Define the relation R
27     R_attributes = ["Name", "Age", "City"]
28     R_tuples = [

```

On the right, the 'Shell' tab shows the output of running the code:

```

Original Relation R:
Name | Age | City
-----
Alice | 25 | New York
Bob | 30 | Los Angeles
Charlie | 35 | Chicago

Projected Relation on ['Name', 'City'] :
Name | City
-----
Alice | New York
Bob | Los Angeles
Charlie | Chicago

```

## 2.

class Relation:

```
def __init__(self, attributes, tuples):
```

```
    self.attributes = attributes
```

```
    self.tuples = tuples
```

```
def cartesian_product(self, other_relation):
```

```
    result_attributes = self.attributes + other_relation.attributes
```

```
    result_tuples = []
```

```
    for tuple1 in self.tuples:
```

```
        for tuple2 in other_relation.tuples:
```

```
            result_tuples.append(tuple1 + tuple2)
```

```
    return Relation(result_attributes, result_tuples)
```

```
def __str__(self):
```

```
    header = " | ".join(self.attributes)
```

```
    lines = [header, "-" * len(header)]
```

```
for tuple in self.tuples:

    lines.append(" | ".join(map(str, tuple)))

return "\n".join(lines)
```

# Example usage

```
if __name__ == "__main__":
```

```
    # Define relation R
```

```
    R_attributes = ["A", "B"]
```

```
    R_tuples = [
```

```
        [1, 2],
```

```
        [3, 4]
```

```
    ]
```

```
    R = Relation(R_attributes, R_tuples)
```

```
    # Define relation S
```

```
    S_attributes = ["C", "D"]
```

```
    S_tuples = [
```

```
        [5, 6],
```

```
        [7, 8]
```

```
    ]
```

```
    S = Relation(S_attributes, S_tuples)
```

```
    # Perform Cartesian Product
```

```
    cartesian_result = R.cartesian_product(S)
```

```
    # Output the resulting relation after the Cartesian Product operation
```

```
    print("Relation R:")
```

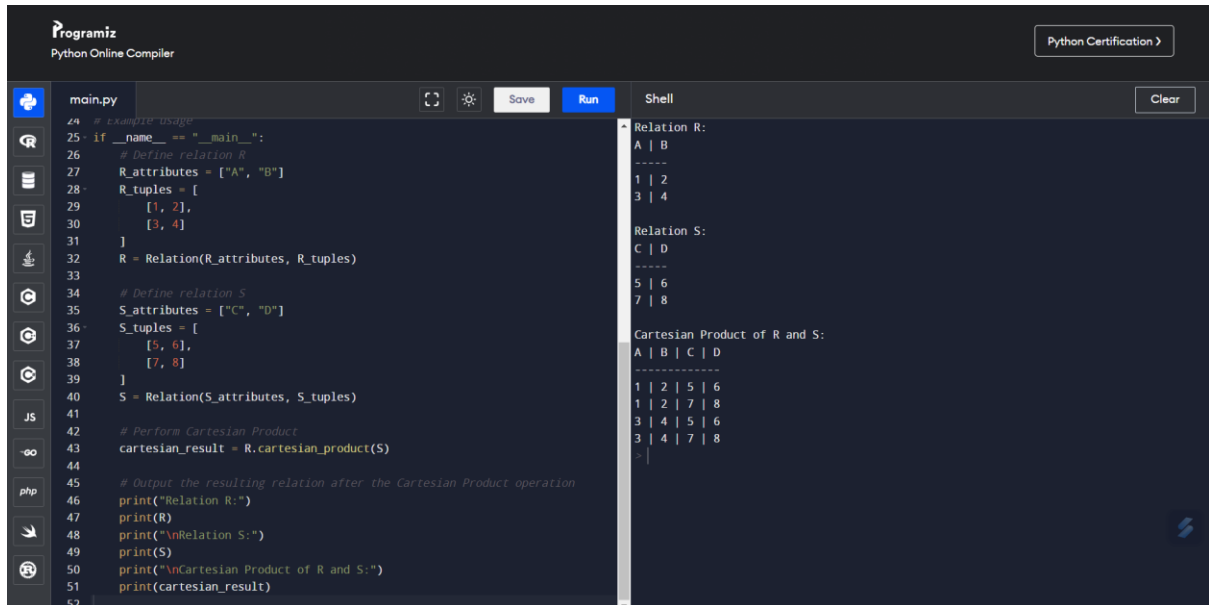
```
    print(R)
```

```
    print("\nRelation S:")
```

```
    print(S)
```

```
print("\nCartesian Product of R and S:")
```

```
print(cartesian_result)
```



The screenshot shows the Programiz Python Online Compiler interface. The left pane displays a Python script in `main.py` that defines two relations, R and S, and calculates their Cartesian product. The right pane shows the output of the script in the Shell.

```
main.py
# Example Usage
25 if __name__ == "__main__":
26     # Define relation R
27     R_attributes = ["A", "B"]
28     R_tuples = [
29         [1, 2],
30         [3, 4]
31     ]
32     R = Relation(R_attributes, R_tuples)
33
34     # Define relation S
35     S_attributes = ["C", "D"]
36     S_tuples = [
37         [5, 6],
38         [7, 8]
39     ]
40     S = Relation(S_attributes, S_tuples)
41
42     # Perform Cartesian Product
43     cartesian_result = R.cartesian_product(S)
44
45     # Output the resulting relation after the Cartesian Product operation
46     print("Relation R:")
47     print(R)
48     print("\nRelation S:")
49     print(S)
50     print("\nCartesian Product of R and S:")
51     print(cartesian_result)
52
```

Shell

```
Relation R:
A | B
----
1 | 2
3 | 4

Relation S:
C | D
----
5 | 6
7 | 8

Cartesian Product of R and S:
A | B | C | D
-----
1 | 2 | 5 | 6
1 | 2 | 7 | 8
3 | 4 | 5 | 6
3 | 4 | 7 | 8
>>
```

3.

class Relation:

```
def __init__(self, attributes, tuples):
```

```
    self.attributes = attributes
```

```
    self.tuples = tuples
```

```
def natural_join(self, other_relation):
```

```
    common_attributes = list(set(self.attributes).intersection(other_relation.attributes))
```

```
    # Find indices of common attributes in each relation
```

```
    self_indices = [self.attributes.index(attr) for attr in common_attributes]
```

```
    other_indices = [other_relation.attributes.index(attr) for attr in common_attributes]
```

```
    # Build the resulting attributes list
```

```
    result_attributes = self.attributes + [attr for attr in other_relation.attributes if attr not in
common_attributes]
```

```

# Build the resulting tuples
result_tuples = []
for tuple1 in self.tuples:
    for tuple2 in other_relation.tuples:
        # Check if tuples match on common attributes
        match = all(tuple1[i] == tuple2[j] for i, j in zip(self_indices, other_indices))
        if match:
            # Combine tuples
            new_tuple = tuple1 + tuple2
            result_tuples.append(new_tuple)

return Relation(result_attributes, result_tuples)

```

```

def __str__(self):
    header = " | ".join(self.attributes)
    lines = [header, "-" * len(header)]
    for tuple in self.tuples:
        lines.append(" | ".join(map(str, tuple)))
    return "\n".join(lines)

```

# Example usage

```

if __name__ == "__main__":
    # Define relation R
    R_attributes = ["A", "B", "C"]
    R_tuples = [
        (1, 2, 3),
        (4, 5, 6)
    ]
    R = Relation(R_attributes, R_tuples)

```

```
# Define relation S
```

```
S_attributes = ["C", "D", "E"]
```

```
S_tuples = [
```

```
    (3, 7, 8),
```

```
    (9, 10, 11)
```

```
]
```

```
S = Relation(S_attributes, S_tuples)
```

```
# Perform Natural Join
```

```
natural_join_result = R.natural_join(S)
```

```
# Output the resulting relation after the Natural Join operation
```

```
print("Relation R:")
```

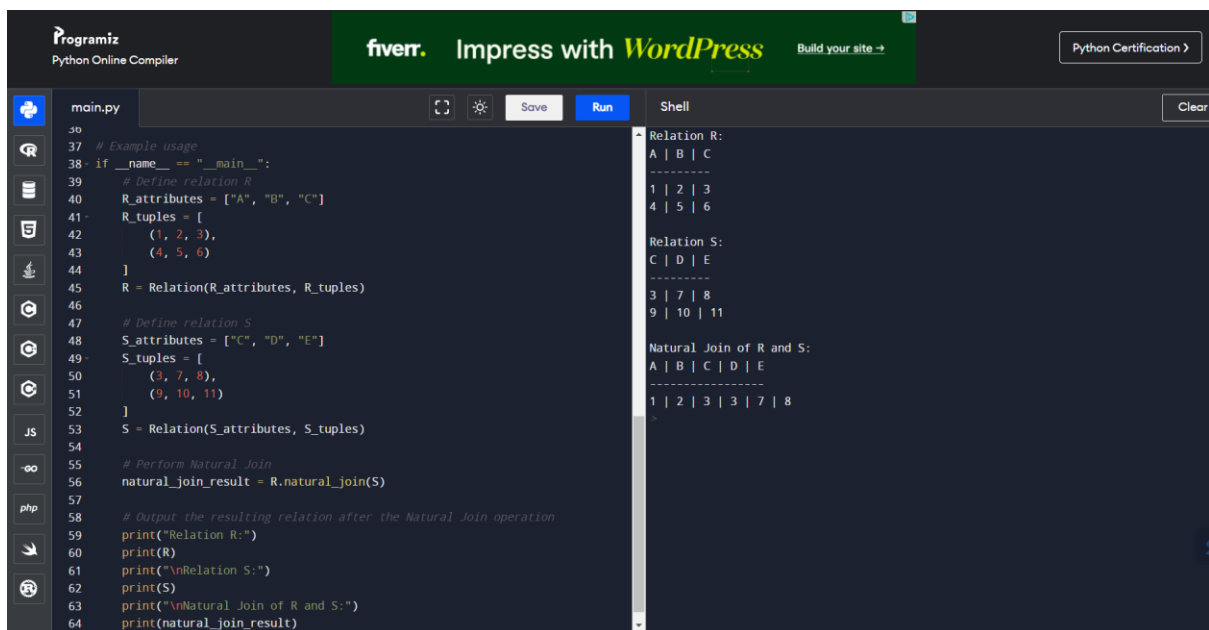
```
print(R)
```

```
print("\nRelation S:")
```

```
print(S)
```

```
print("\nNatural Join of R and S:")
```

```
print(natural_join_result)
```



```
main.py 30 37 # Example usage 38 if __name__ == "__main__": 39 # Define relation R 40 R_attributes = ["A", "B", "C"] 41 R_tuples = [ 42 (1, 2, 3), 43 (4, 5, 6) 44 ] 45 R = Relation(R_attributes, R_tuples) 46 47 # Define relation S 48 S_attributes = ["C", "D", "E"] 49 S_tuples = [ 50 (3, 7, 8), 51 (9, 10, 11) 52 ] 53 S = Relation(S_attributes, S_tuples) 54 55 # Perform Natural Join 56 natural_join_result = R.natural_join(S) 57 58 # Output the resulting relation after the Natural Join operation 59 print("Relation R:") 60 print(R) 61 print("\nRelation S:") 62 print(S) 63 print("\nNatural Join of R and S:") 64 print(natural_join_result)
```

Shell

```
Relation R:  
A | B | C  
-----  
1 | 2 | 3  
4 | 5 | 6  
  
Relation S:  
C | D | E  
-----  
3 | 7 | 8  
9 | 10 | 11  
  
Natural Join of R and S:  
A | B | C | D | E  
-----  
1 | 2 | 3 | 3 | 7 | 8
```

4.

class Relation:

```
def __init__(self, attributes, tuples):
```

```
    self.attributes = attributes
```

```
    self.tuples = tuples
```

```
def division(self, other_relation):
```

```
    # Check if the attributes of S are a subset of the attributes of R
```

```
    if not set(other_relation.attributes).issubset(set(self.attributes)):
```

```
        print("Error: The attributes of S are not a subset of the attributes of R.")
```

```
        return None
```

```
    # Find the indices of attributes in R that are not in S
```

```
    indices_to_keep = [self.attributes.index(attr) for attr in self.attributes if attr not in  
other_relation.attributes]
```

```
    # Build the resulting attributes list
```

```
    result_attributes = [attr for attr in self.attributes if attr not in other_relation.attributes]
```

```
    # Build the resulting tuples
```

```
    result_tuples = []
```

```
    for tuple1 in self.tuples:
```

```
        match_found = False
```

```
        for tuple2 in other_relation.tuples:
```

```
            # Check if tuple2 is a subset of tuple1
```

```
            if all(tuple1[i] == tuple2[other_relation.attributes.index(attr)] for i, attr in  
enumerate(other_relation.attributes)):
```

```
                match_found = True
```

```
                break
```

```
        if not match_found:
```

```
            result_tuples.append([tuple1[i] for i in indices_to_keep])
```



```
return Relation(result_attributes, result_tuples)
```

```
def __str__(self):  
    header = " | ".join(self.attributes)  
    lines = [header, "-" * len(header)]  
    for tuple in self.tuples:  
        lines.append(" | ".join(map(str, tuple)))  
    return "\n".join(lines)
```

```
# Example usage
```

```
if __name__ == "__main__":
```

```
    # Define relation R
```

```
    R_attributes = ["A", "B"]
```

```
    R_tuples = [  
        (1, 2),  
        (3, 4),  
        (5, 6)  
    ]
```

```
    R = Relation(R_attributes, R_tuples)
```

```
    # Define relation S
```

```
    S_attributes = ["A"]
```

```
    S_tuples = [  
        (1,),  
        (3,)   
    ]
```

```
    S = Relation(S_attributes, S_tuples)
```

```
    # Perform Division ( $R \div S$ )
```

```
division_result = R.division(S)
```

```
# Output the resulting relation after the Division operation
```

```
print("Relation R:")
```

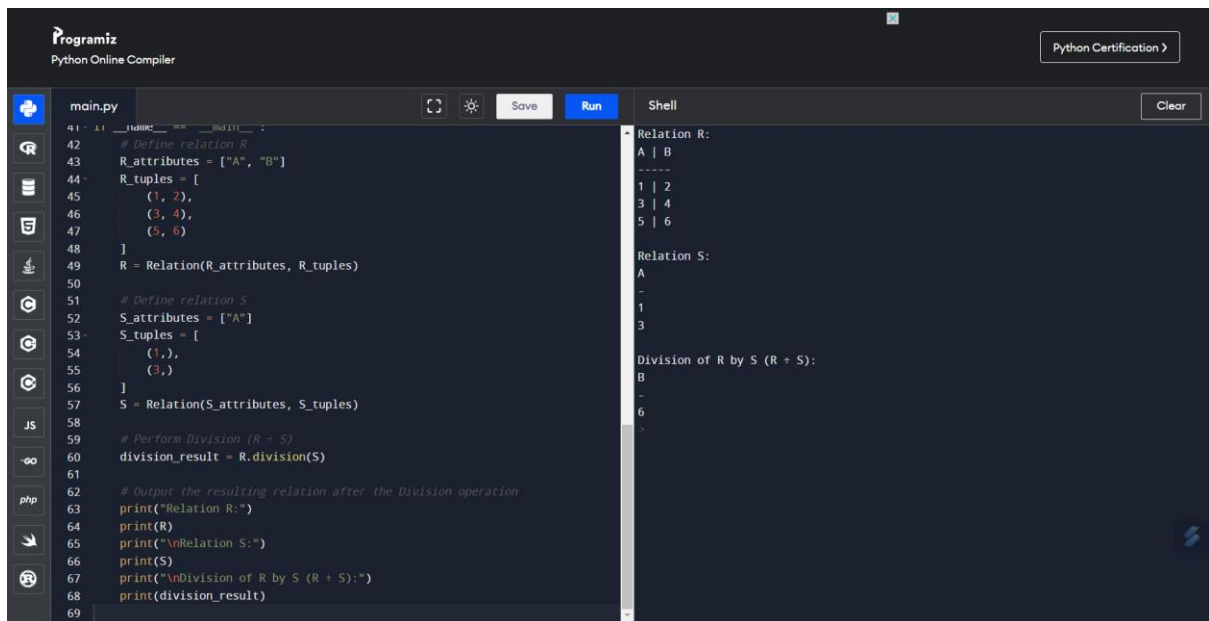
```
print(R)
```

```
print("\nRelation S:")
```

```
print(S)
```

```
print("\nDivision of R by S (R ÷ S):")
```

```
print(ddivision_result)
```



The screenshot shows the Programiz Python Online Compiler interface. The left sidebar contains icons for various programming languages: Python, JavaScript, PHP, and others. The main editor displays a Python script named `main.py` with the following code:

```
41 # Define relation R
42 R_attributes = ["A", "B"]
43 R_tuples = [
44     (1, 2),
45     (3, 4),
46     (5, 6)
47 ]
48 R = Relation(R_attributes, R_tuples)
49
50 # Define relation S
51 S_attributes = ["A"]
52 S_tuples = [
53     (1,),
54     (3,)
55 ]
56 S = Relation(S_attributes, S_tuples)
57
58 # Perform Division (R ÷ S)
59 division_result = R.division(S)
60
61 # Output the resulting relation after the Division operation
62 print("Relation R:")
63 print(R)
64 print("\nRelation S:")
65 print(S)
66 print("\nDivision of R by S (R ÷ S):")
67 print(ddivision_result)
```

The right sidebar shows the output of the script in the Shell:

```
Relation R:
A | B
---
1 | 2
3 | 4
5 | 6

Relation S:
A
-
1
3

Division of R by S (R ÷ S):
B
-
6
```